Escuela
Politécnica
Superior

Unicar App

Aplicación web donde compartir coche para ir a la universidad

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Héctor Esteve Yagüe
Tutor/es:
Javier Montoyo Bojo

Universitat d'Alacant
Universidad de Alicante

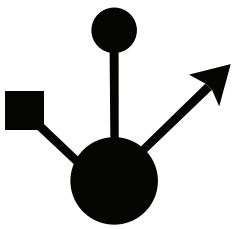
Julio 2021

Unicar App

Aplicación web donde compartir coche para ir a la universidad

Autor
Héctor Esteve Yagüe

Tutor/es
Javier Montoyo Bojo
CIENCIA DE LA COMPUTACION E INTELIGENCIA ARTICIAL.



Grado en Ingeniería Multimedia

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

Resumen

El desplazamiento a la universidad es uno de los aspectos básicos a tener en cuenta durante la vida universitaria. Para ello, cada vez resulta más interesante la idea de compartir coche en lugar de emplear el transporte público o el transporte privado e individual.

Por lo que respecta a estas formas tradicionales, el transporte público es, generalmente, carente de flexibilidad horaria para un universitario, mientras que el transporte privado individual puede provocar aglomeraciones en el campus. Así pues, pueden ser muchos los motivos por los que a un universitario no le resulte cómodo desplazarse a la universidad.

Con el objetivo de solventar esta preocupación, *Unicar App* surge como alternativa a estos tipos de transporte, permitiendo a los universitarios encontrar a otros universitarios con horarios similares para ir juntos a la universidad.

Esta práctica de compartir vehículo, conocida recientemente por el concepto 'carpooling' (de la suma de las palabras en inglés 'Car' y 'Pooling', en español 'Coche' y 'Agrupación'), es cada vez más popular y conlleva muchos beneficios. Entre ellos destacan, principalmente, el ahorro de gastos por combustible y la reducción del número de vehículos en las vías. No obstante, a ello se le puede añadir un tercer aspecto social, como puede ser el hecho de establecer nuevas relaciones.

Si bien es cierto que en el mercado existen aplicaciones similares para viajar compartiendo coche, en *Unicar App*, más allá de publicar y buscar viajes, es posible planificar semanalmente estos desplazamientos, ofreciendo incluso la posibilidad de añadir el calendario particular de clases garantizando así una óptima organización ajustada a las necesidades del usuario. Esto, además, posibilita a la aplicación obtener coincidencias entre los horarios de los usuarios, ofreciendo así sugerencias de usuarios con quien compartir viaje.

En definitiva, *Unicar App* es mucho más que una aplicación para viajar en conjunto. A su vez, se trata de un planificador de desplazamientos diarios en el que el hecho de ir a la universidad dejará de ser un quebradero de cabeza.

Abstract

Nowadays, going to university is one of the main aspects to bear in mind during the university period. Instead of using the public transport or the private and individual transport, sharing the car is eventually becoming more attractive.

Public transport services are not usually flexible enough for a student, and the private transport could cause crowds at the campus. Hence, there are a lot of reasons for which a university student does not find it comfortable to go to university.

In order to solve this concern, *Unicar App* emerges as an alternative to make these displacements easier, allowing university students to find other ones with similar schedule to go together.

Sharing the car, recently known as 'carpooling' (merging from 'Car' and 'Pooling'), is getting more popular and implies lots of benefits. Saving money for the fuel and minimising the number of cars on the tracks are prominent among them. In addition, we may take into account other social aspect, as it is meeting new people.

Despite the fact that there are similar applications in the market to share the car, in *Unicar App*, apart from publishing and searching journeys, it is possible to make a weekly planning for these displacements. In addition, *Unicar App* offers its users the possibility of adding their particular calendar, allowing the application to find similar schedules and suggesting it to them.

In conclusion, *Unicar App* is much more than an application for travelling in group. It is also a daily planner in which going to the university is not a headache anymore.

Resum

Desplaçar-se a la universitat és un dels aspectes bàsics a tindre en compte durant la vida universitària. Per fer-ho, cada vegada resulta més interesant la idea de compartir cotxe en lloc de fer ús del transport públic o el transport privat i individual.

Pel que respecta a aquestes formes tradicionals, el transport públic és, generalment, faltat de flexibilitat horària per a un universitari, mentre que el transport privat individual pot provocar aglomeracions al campus. Així doncs, poden ser molts els motius pels quals a un universitari no li resulte còmode desplaçar-se a la universitat.

Amb l'objectiu de solventar aquesta preocupació, *Unicar App* surt com alternativa a aquests tipus de transport, permetent als universitaris encontrar altres universitaris amb horaris similars per anar juntament a la universitat.

Aquesta pràctica de compartir vehicle, coneguda recentment pel concepte 'carpooling' (de la suma de les paraules en anglés 'Car' i 'Pooling', en valencià Cotxe i Agrupació), és cada vegada més popular i comporta molts beneficis. Entre ells destaquen principalment l'estalvi de despeses per combustible i la reducció del nombre de vehícles a les vies. No obstant això, a aquests fets se li pot afegir un tercer aspecte social, com pot ser el fet d'establir noves relacions.

Si bé és cert que al mercat existeixen aplicacions similars per viatjar compartint cotxe, a *Unicar App*, més enllà de publicar i buscar viatjes, és possible planificar semanalment aquests desplaçaments. Fins i tot oferint la possibilitat d'afegir el calendari particular de classes garantitzant així una òptima organització ajustada a les necessitats de cada usuari. Açò, a més a més, possibilita a l'aplicació obtindre coincidències entre els horaris dels usuaris, oferint així subgerències d'usuaris amb qui compartir viatje.

En definitiva, *Unicar App* és molt més que una aplicació per viatjar en conjunt. Al seu torn, es tracta d'un planificador de desplaçaments diaris en el que el fet de anar a la universitat deixarà de ser un maldecap.

Motivación, justificación y objetivo general

Para comenzar con este documento, voy a explicar el contexto en el que me encontraba cuando surgió la idea de crear una aplicación para gestionar el transporte a la universidad compartiendo vehículo propio.

Mi residencia habitual es Castalla, un pueblo situado en la provincia de Alicante, concretamente en la comarca de "La Foia de Castalla". Se trata de una comarca del interior de la provincia, a aproximadamente 25 minutos en coche de la Universidad. Este tiempo no lo he considerado excesivo durante toda mi etapa universitaria, por lo que la idea de vivir en una residencia o piso universitario nunca ha sido una alternativa. Por esta razón he tenido que desplazarme todos los días desde mi pueblo hasta la universidad probando todas las maneras posibles.

Así pues, me remonto a justo un mes antes de empezar las clases de mi primer curso del grado. No disponía de carnet de coche y por tanto debía buscar un medio de transporte con el que desplazarme cada día.

La primera alternativa aparente fue desplazarme con autobús. Para mi sorpresa, únicamente había una única manera de llegar desde Castalla a la universidad con autobús, y no está enfocada para los universitarios. Por razones que expondré más adelante, esta siempre me ha resultado la menos oportuna y por tanto desde el primer momento pensé que de alguna manera debían estar organizándose los universitarios de la zona con tal de poder ir a la universidad sin este autobús.

Poco después supe que había un grupo de WhatsApp que reunía a una parte de los universitarios del pueblo, todos ellos, eso sí, estudiantes de la UA. Cuando entré a este grupo, ya contaba con varios años de existencia y desde entonces ha ido creciendo mucho con el paso del tiempo. Ha crecido hasta tal punto que la inmensa mayoría de estudiantes de la UA del pueblo están en él. La funcionalidad es muy básica, simplemente se trata de una especie de tablero de anuncios donde encontramos con quien compartir coche. Sin embargo, desde que entré no siempre ha funcionado así. Tal y como explicaré más adelante, el funcionamiento ha ido evolucionando, y no siempre con mejoras.

Después de 4 años esta ha sido la manera más oportuna para desplazarme, bien compartiendo mi coche o bien con algún compañero. Por la experiencia analizando el funcionamiento y la idea que ya me rondaba pocas semanas después de entrar al grupo, veo en el TFG la oportunidad de desarrollar una aplicación que minimice los problemas que he ido observando.

Por esta razón decidí dedicar mi TFG a solventar este problema ofreciendo a los estudiantes de la zona donde resido, y en un futuro a todos aquellos que compartan este problema, una forma fácil y eficaz de encontrar personas con las que compartir coche para ir a la universidad.

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor Javier por su disposición durante estos meses. Ha sido un placer trabajar bajo su tutoría, por sus consejos y. especialmente, por el interés que ha demostrado en mejorar cada detalle de este proyecto.

También quiero agradecer a los profesores de Ingeniería Multimedia que durante estos años me han hecho crecer académicamente. Y a mis compañeros, de los que me llevo, además de amistades, una magnífica experiencia trabajando junto a ellos.

Por último, quisiera dar las gracias a mi grupo de amigos, por estar en todo momento a mi lado. A mi pareja, por su comprensión, ánimo, fuerza y colaboración tan necesarias como reconfortantes. Y especialmente a mi familia, que desde el momento en que mi madre nos dejó, han sido el mejor refugio para afrontar este golpe tan duro.

Y a ella, de la que no tengo ninguna duda de que ha sido la principal razón por la que todo esto ha sido posible. Porque no puedo estar más agradecido de que seas mi ejemplo a seguir y por poder dedicarte este trabajo del que seguro estarás orgullosa. *Gràcies*

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

Tal y como he comentado en el apartado de Motivación, el objetivo de este TFG es obtener como resultado final una aplicación que pueda ofrecer una alternativa a la manera actual en que los estudiantes de la zona donde resido puedan encontrar con quien compartir coche. TODO. Problema para todos pero se probará inicialmente en Castalla.... Considero que tanto la oferta que hay con respecto al medio de transporte público (autobús) como la organización que actualmente hay en el mencionado grupo, no resulta suficientemente cómoda para el día a día, sino que conlleva más de un quebradero de cabeza.

Por lo que respecta a la alternativa del medio de transporte público, se trata de un autobús común que parte de Alcoy con dirección a Alicante y pasa por los diferentes pueblos de la zona. Me enteré de que, por petición popular, a la ruta que hace este autobús se le añadió como parada una situada en San Vicente, que por proximidad valdría para ir a la Universidad. Esta es la primera razón por la que no podemos considerar que esta vía no está expresamente destinada a estudiantes, pues el paso por San Vicente se introdujo a modo de parche.

Dejando esta razón como anecdótica, ya que al fin y al cabo el autobús tiene una parada cerca de la Universidad, los dos motivos que más inconvenientes suponen son los siguientes.

En primer lugar, un horario carente de variedad y poco fiable. Este no está planteado para los más que diversos horarios de la Universidad. Para ir, con 3 pasos repartidos durante la mañana y 3 repartidos por la tarde, es complicado encontrar uno de ellos con el que llegar a clase sin excesiva anterioridad o sin retraso. El horario actual está reflejado en la figura 1.1. Además de esto, por experiencia, las horas establecidas para los pasos no son fiables. A pesar de estar avisado que la hora de paso puede oscilar unos 5 minutos, he llegado a experimentar adelantos y retrasos de 10-15 minutos. Esto supone perder el autobús si pasa antes de lo previsto o llegar tarde en caso de retraso.

	Ida >										
Alcoi	6:15	-	11:00	-	13:00	14:30	14:30	17:30	20:15	-	-
Ibi	6:45	8:00	11:30	-	13:30	15:00	15:00	18:00	20:45	-	-
Onil	7:00	8:15	11:45	-	13:45	15:15	15:15	18:15	21:00	-	-
Castalla	7:15	8:30	12:00	-	14:00	15:30	15:30	18:30	21:15	-	-
Biar	-	-	-	-	-	-	TAD	-	-	-	-
Villena	-	-	-	-	-	-	TAD	-	-	-	-
Tibi	TAD	-	-	-	-	-	-	-	-	-	-
Maigmo	7:25	8:40	12:10	-	14:10	15:40	-	18:40	21:25	-	-
San Vicente	7:40	8:55	12:25	-	14:25	15:55	-	18:55	21:40	-	-
Alicante - Estación de Autobuses	8:00	-	12:45	-	14:45	16:15	-	19:15	22:00	-	-

Figura 1.1: Horario de ida del autobús, resaltados los pasos por Castalla y San Vicente
Fuente propia

El otro inconveniente es el precio, excesivamente alto. También por no estar destinado a universitarios, el viaje en el caso de subir en Castalla hasta la San Vicente es de aproximadamente 4,50€. Este es un precio inasumible teniendo en cuenta que ir y volver supondría un gasto de aproximadamente 9€ diario. El cálculo es rápido: 9€ diarios, 5 días a la semana y 4 semanas por mes, suponen 180€ al mes. No obstante, en los últimos años, también a petición popular, se introdujo un bono que reduce considerablemente este gasto. Se trata de un bono de 10 viajes por 25€. Esto es, 2 viajes cada día implican 10 viajes a la semana, por lo que serían 25€ semanales, un total 100€ al mes. En definitiva, continúa siendo un precio muy elevado.

Por la suma de estas razones podemos considerar que el autobús no es una solución apetecible al problema que supone el desplazamiento de los universitarios residentes en esta zona.

La alternativa a este transporte público es el transporte privado, y dentro de este, con el fin de ahorrar gastos y además contribuir favorablemente con el medio ambiente, es una buena conducta compartir coche con personas que tengan un trayecto y horario similar. Con estos principios surgió la idea del grupo de WhatsApp.

Me comentaron que, en sus inicios, contaba con poca gente en comparación a la actual y entonces la organización era bastante sencilla y efectiva. Se pasaban los horarios del cuatrimestre y un encargado realizaba una tabla con la distribución de coches según el horario de

cada persona. De esta manera se podían distribuir las personas con horarios similares para compartir coche y además alternar el coche con el que se viajaba, para no conducir todos los días las mismas personas. Un ejemplo de cómo se realizaba esta distribución está en la figura 1.2.

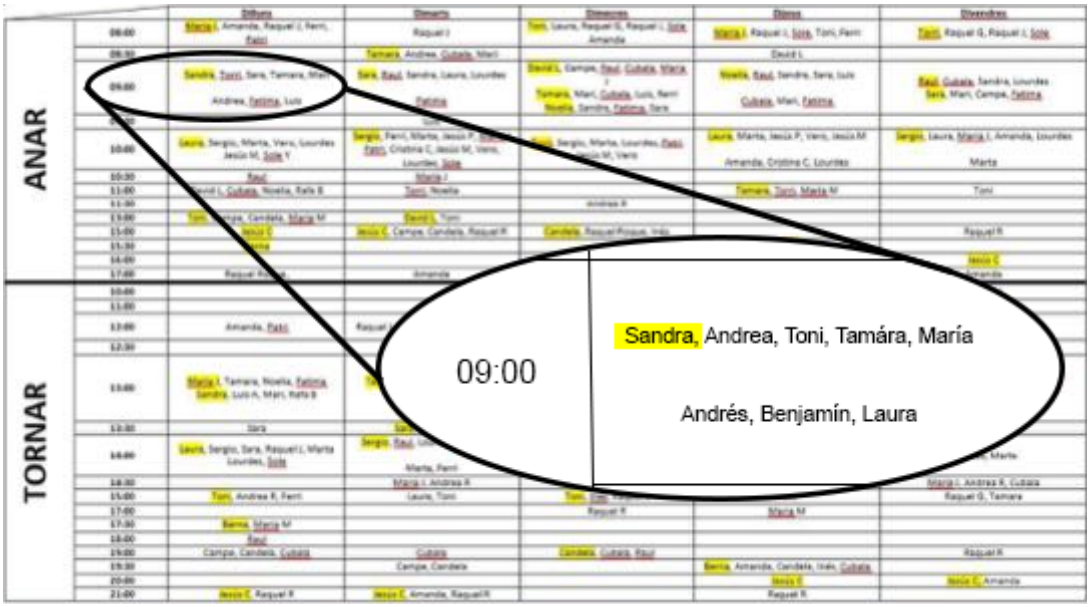


Figura 1.2: Ejemplo de horario con distribución de coches, en amarillo el conductor
Fuente propia

Cuando entré yo a este grupo, continuaba vigente esta organización. Sin embargo, el número de personas había crecido considerablemente y el trabajo que suponía recoger todos los horarios y compatibilizarlos resultaba ya muy engorroso. Surgieron más inconvenientes, ya que había personas que, estando en el grupo, no querían estar obligados a cumplir un horario, y les resultaba más cómodo adaptarse según les convenía al que publicaba el encargado con los pocos horarios que le pasaban. Otras personas, sin embargo, no podían pasar un horario para el cuatrimestre ya que cada semana era variable. En resumen, por una razón o por otra, esta manera de organizarse ya no resultaba efectiva para todos y el encargado desistió en su intento de hacer el horario.

Con el paso del tiempo y cada vez menos integrantes siguiendo el horario, la forma de organizar los viajes ha cambiado. El grupo, con más de 120 universitarios, sirve como tablón de anuncios en el que hay 2 tipos de interacción: la persona que anuncia que va a ir a cierta hora a la universidad y la persona que pregunta si alguien va a ir a cierta hora a la universidad. En la figura 1.3 podemos observar ejemplos de estas interacciones. Por razones obvias, la docencia telemática ha paralizado temporalmente la actividad del grupo, que era muy concurrente. Queda por ver, a día de hoy, la evolución que tiene la pandemia y los efectos que esta puede provocar en aspectos como compartir coche con diferentes personas cada día. A pesar de ello es de sentido común esperar y desear que vuelva en cierta manera

la normalidad a la que estábamos acostumbrados.

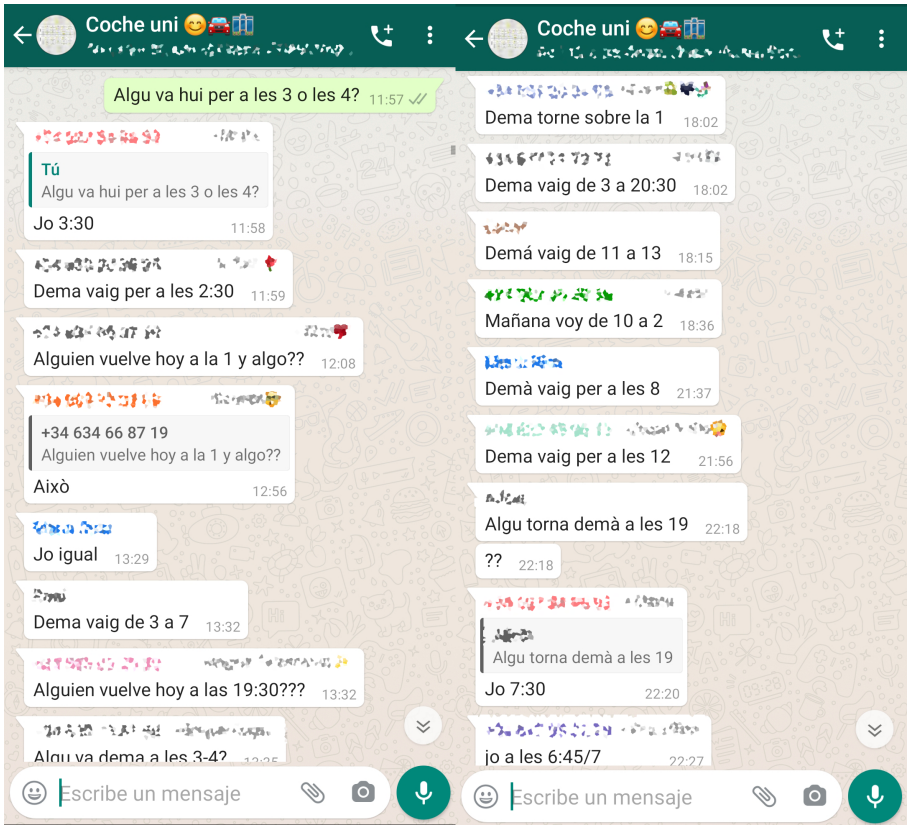


Figura 1.3: Ejemplos de interacciones en el grupo.
Fuente propia

Una vez contextualizada la problemática, la solución que llevo años meditando es crear una plataforma digital. Será una aplicación que permitirá a aquellos estudiantes que no les resulte cómodo desplazarse desde su residencia habitual encontrar a otros que ofrezcan una plaza en su coche para ir juntos. Para facilitar la organización tendrá como referencias las diferentes formas en que ha funcionado el grupo de WhatsApp, siendo esta aplicación un sustitutivo que además de cumplir la misma función mejore la experiencia del usuario.

2. Estudio de la viabilidad

2.1. Análisis DAFO

El análisis DAFO se trata de una herramienta que da como resultado un esquema en el que se determina la situación de la idea. Para aplicarlo, se distinguen 2 análisis de esta situación: el interno y el externo, y dentro de cada uno de ellos se valoran los aspectos favorables y los desfavorables. Las siglas DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades) nos indican los 4 aspectos a valorar.

	Análisis interno	Análisis externo
Aspectos desfavorables	• Desconocimiento de alguna tecnología a emplear.	• Rechazo inicial del público objetivo a novedades si ya existe algo que cumple con las exigencias.
Aspectos favorables	• 4 años de experiencia en el ámbito. • Conocimiento y experiencia con gran parte de las tecnologías a emplear.	• Evolución constante en la organización en el grupo. • Descenso de la actividad en el grupo. • Público objetivo muy definido y acostumbrado a usar aplicaciones para todo.

Figura 2.1: Análisis DAFO aplicado a la aplicación
Fuente propia

2.2. Lean Canvas

El Lean Canvas también es una herramienta que, al aplicarla sobre la idea, da como resultado un esquema. En este caso ya se tratan aspectos más concretos que resultarán útiles para mejorar la idea y el esquema resultado expondrá el modelo de negocio. En la figura 2.2 podemos observar el resultado obtenido al plasmar en el canvas modelo de negocio. Pero antes, se exponen detalladamente los bloques aplicados al modelo de negocio en cuestión que

conforman este gráfico, que son:

- **Segmentos de clientes:**

  Como clientes distinguimos dos tipos. En primer lugar, los clientes inciales, que serían el primer grupo de usuarios al que está enfocada la aplicación. Se trata de los estudiantes de la zona donde resido, es decir, aquellos que comparten el mismo inconveniente que yo para desplazarse a la universidad. El otro grupo a considerar, los potenciales, serían aquellos que en un futuro podrían hacer uso de la aplicación si esta se expande a otros niveles, como sería aplicarlo a diferentes zonas o incluso a diferentes universidades.

- **Problemas:**

  Los principales problemas a los que daría solución la aplicación son cuatro. El principal es lo dificultoso que puede resultar encontrar una forma de desplazarse a la universidad cuando el transporte público no lo cubre con suficiencia. Siendo el grupo de WhatsApp para compartir coche la mejor alternativa, en ella encontramos los otros tres inconvenientes. En el lado de la persona que busca coche, la deficiente organización actual del grupo que no permite tener estabilidad en cuanto a horario, lo que provoca que prácticamente cada día sea necesario buscar con quien ir al día siguiente. Esta búsqueda resulta bastante tediosa e incluso puede llegar a ser fallida si no se encuentra. Y finalmente, por lo que respecta a la persona ofrece plaza en su coche, se han dado casos de contar con más personas de las que pueden subir, viéndose en la obligación de dejar a alguno de los pasajeros sin poder ir.

- **Proposición de valor única:**

  La aplicación, con la intención de solventar los problemas descritos, permitirá gestionar el desplazamiento de los estudiantes a la universidad de forma personalizada (a partir del calendario y el horario), siendo el principal objetivo fomentar el hecho de compartir vehículo particular con los beneficios que esto conlleva.

- **Solución:**

  A grandes rasgos, las soluciones son tres. Cada usuario podrá importar su horario y calendario para tener control de cada desplazamiento que deba realizar. Además, la aplicación ofrecerá un servicio de búsqueda que por medio de filtros permitirá encontrar ofertas de otros usuarios que compartan su coche. Si los viajes compartidos son recurrentes, se podrá establecer una planificación periódica de estos.

- **Canales:**

  El canal más directo para llegar al primer nicho de clientes es evidente, pues por medio del propio grupo de WhatsApp se puede llegar a todos y cada uno de ellos. De cara a futuro, es interesante difundir la marca por medio de redes sociales con fin de llegar al

prototipo de usuario joven y estudiante, además de una web corporativa donde publicar constantemente contenido relevante sobre la aplicación y actualizaciones.

- **Flujos de ingresos:**

  Con respecto a los ingresos previstos, inicialmente estos vendrían por medio de donaciones para cubrir los costes de mantenimiento. La organización actual del grupo de WhatsApp tiene estipulado un aportación económica al conductor para repartir los gastos del vehículo, por lo que no seria una idea muy disparatada pedir una pequeña donación a los usuarios. De cara a futuro, esta fuente de ingresos debería de ser sustituida por una más comercial.

- **Estructura de costes:**

  Los costes iniciales que supone la puesta en producción de la aplicación es contratar un servidor donde alojarla y su respectivo dominio.

- **Métricas clave:**

  Los indicadores que nos permitirán ver la evolución del proyecto serán, entre otros, el número de estudiantes de la zona que hacen uso de la aplicación y el número de viajes que se concretan por medio de esta.

- **Ventaja diferencial:**

  Como ventaja diferencial tengo la experiencia de haber estado 4 años en el grupo al que hago referencia y desde las pocas semanas de estar dentro, he estado pensando la manera de optimizar su organización. Además, a diferencia de otras aplicaciones similares para compartir coche, esta tendrá un enfoque universitario.

<table>
<tr>
<td rowspan="2">Problemas

Dificultad para encontrar forma de desplazamiento a universidad.

Deficiente organización del grupo en la actualidad.

Búsqueda tediosa y no siempre efectiva en el grupo.

Posible descontrol respecto a número de pasajeros.</td>
<td>Soluciones

Horarios y calendarios personalizables.

Búsqueda clara de coches disponibles para ir a cierta hora.

Planificación periódica de los viajes.</td>
<td rowspan="2">Propuesta de valor

Gestión del desplazamiento a la universidad de forma personalizada y basado en el hecho de compartir vehículo particular.</td>
<td>Ventaja diferencial

4 años de experiencia en un grupo compartiendo coche.

Amplio conocimiento de la causa.

Enfoque universitario.</td>
<td rowspan="2">Segmentos de clientes

Iniciales: Estudiantes de la UA residentes en “La Foia de Castalla”.

Potenciales: Estudiantes sin residencia cercana a sus universidades.</td>
</tr>
<tr>
<td>Métricas clave

Número de estudiantes de la zona registrados.

Número de viajes que se realizan desde la zona hasta la universidad y viceversa.</td>
<td>Canales

Grupo de Whats-App actual.</td>
</tr>
<tr>
<td colspan="2">Estructura de costes

Costes de hosting y dominio.</td>
<td colspan="3">Flujo de ingresos

Inicialmente: donaciones para cubrir gastos de mantenimiento.</td>
</tr>
</table>

Figura 2.2: Modelo de negocio con Lean Canvas

2.3. Análisis de riesgos durante el desarrollo

En este apartado serán definidos los principales riesgos a los cuales estaría expuesto durante el desarrollo del TFG. Cada uno de ellos tendrá una valoración por intervalos de la probabilidad y los efectos que estos tendrían, además del plan de prevención y contingencia para cada uno de ellos. Los riesgos están distinguidos entre tres tipos: personales, de planificación y tecnológicos.

Estos son los intervalos de valoración aplicados:

- Probabilidad: Muy Baja (<10%), Baja (10-25%), Moderada (25-50%), Alta (50-75%) y Muy Alta (>75%)
- Efectos: Catastrófico, Serio, Tolerable o Insignificante

2.3.1. Riesgos personales

Estos son aquellos riesgos ajenos al desarrollo del TFG pero que, por afectarme personalmente, tienen una repercusión directa.

- Enfermedad:

  Por desgracia, en la situación mundial no es extraño ver que prácticamente cualquier persona puede caer enferma. Obviamente la referencia es sobre el COVID-19. Sus efectos son muy diversos e imprevisibles, y por tanto es necesario ser consciente de en caso de caer enfermo podría obligar a parar el desarrollo del TFG. En mi caso, tras haber sido positivo hace unos meses, la probabilidad es muy baja pero igual de valorable.

  Probabilidad: Muy baja

  Efecto: Serio

  Plan de prevención: Seguir las restricciones aplicadas por las autoridades.

  Plan de contingencia: En caso de caer enfermo, los efectos son tan variables que pueden ser insignificantes sin obstaculizar el desarrollo, hasta catastróficos si por estado grave de salud resulta imposible continuar el trabajo.

- Enfermedad de un familiar:

  Si caer uno mismo enfermo afecta directamente, también lo puede hacer que caiga un familiar. Hago mención especial a este riesgo porque ya lo he sufrido. No ha sido durante el desarrollo, pero ha afectado mis planes respecto al inicio del desarrollo del

TFG. Además de dar yo positivo en COVID-19, mis padres también lo sufrieron con fatal desenlace para mi madre. Con esto, la fecha prevista para el inicio se retrasó prácticamente 2 meses.

Probabilidad: Muy baja

Efecto: Catastrófico (valorando caso propio)

Plan de prevención: Seguir las restricciones aplicadas por las autoridades.

Plan de contingencia: Dado el caso mencionado, sacar el valor necesario para que afecte de la menor manera posible y utilizar los momentos dedicados al TFG como distracción.

- Carga extra de trabajo:

A pesar de no estar cursando ninguna asignatura en este cuatrimestre, me encuentro actualmente realizando las prácticas externas. El horario de 6 horas (más aprox. 1,5 de desplazamiento diario) no me permite mucho tiempo entre semana para dedicar al TFG, por lo que debe ser optimizado. Esto puede verse aún más afectado a lo largo de los meses si en cualquiera de los dos focos de trabajo es necesario un sobre-esfuerzo que conlleve a un estado de saturación.

Probabilidad: Moderada

Efecto: Tolerable

Plan de prevención: Aprovechar óptimamente sacando máximo rendimiento durante el tiempo de desarrollo del TFG y descansar adecuadamente cada día.

Plan de contingencia: En caso de llegar a un punto de saturación, disminuir drásticamente el tiempo dedicado al TFG durante 1 o 2 días con el fin de restaurar el estado de ánimo.

2.3.2. Riesgos de planificación

Estos son los riesgos que pueden conllevar una errónea planificación y que afectarían al tiempo estimado previamente a cada tarea.

- Fechas muy ajustadas:

Por lo que respecta al tiempo disponible hasta la fecha de entrega no es demasiado holgado. El hecho de empezar con retraso aumenta la probabilidad de que no disponer

del tiempo suficiente.

Probabilidad: Moderada

Efecto: Serio

Plan de prevención: Realizar una buena planificación y aprovechar al máximo el tiempo dedicado.

Plan de contingencia: En caso de observar un lento avance y prever un retraso respecto a la fecha de la convocatoria deseada, volver a realizar una estimación de los tiempos de cara a la siguiente convocatoria.

- Mal cálculo de fechas en la planificación:

  La planificación descrita en el siguiente apartado, está hecha, obviamente, sin tener experiencia previa en desarrollar un TFG. Es por ello que para la estimación del desarrollo de la memoria han sido considerados los tiempos recomendados por José Vicente Berna, jefe de estudios del Grado en Ingeniería Multimedia en la Universidad de Alicante, como más adelante detallaré. Sin embargo, estos han sido adaptados al tiempo que dispongo y por tanto puede haber alguna inconsistencia.

  Probabilidad: Baja

  Efecto: Serio

  Plan de prevención: Dedicar suficiente tiempo a valorar detenidamente los diferentes apartados que componen la memoria y las etapas de implementación.

  Plan de contingencia: No continuar con el desarrollo sin antes re-planificar los tiempos valorando qué aspectos previstos son más prioritarios para dedicarles la mayor parte del tiempo.

2.3.3. Riesgos tecnológicos

Este tipo de riesgos son los que pueden surgir durante la etapa de implementación. Hacen referencia mayoritariamente a las tecnologías a emplear.

- Pérdida o corrupción de archivo de la memoria:

  Este documento, el de la memoria, debe ser el más controlado durante el desarrollo. La pérdida de este supondría una pérdida de tiempo enorme y significaría echar por la borda la mayor parte de las horas dedicadas. No obstante, cabe decir que todo lo

discurrido e investigado sería recordable a grandes rasgos y no se partiría desde 0 en un hipotético nuevo comienzo.

Probabilidad: Muy baja

Efecto: Catastrófico

Plan de prevención: Realizar constantemente copias de seguridad de este y en diferentes ubicaciones (p. ej. Ordenador propio, GitHub, Drive, Dropbox, etc.).

Plan de contingencia: En el caso más leve (pérdida en el ordenador habitual) recuperar el archivo desde uno de los repositorios o ubicaciones mencionados. En el peor de los casos, (pérdida total), volver a empezar e intentar recordar a grandes rasgos el contenido de cada uno de los apartados.

- Desconocimiento de la tecnología:

A pesar de tener pensadas las tecnologías a emplear para la aplicación y haber comprobado, sin entrar mucho en detalle, que es posible implementar las funcionalidades deseadas con estas, cabe la posibilidad de que por falta de experiencia con alguna de ellas surjan inconvenientes. Podría pasar que, tras estar un tiempo aprendiendo una tecnología para cierta funcionalidad, esta no termine de ser válida.

Probabilidad: Moderada

Efecto: Insignificante

Plan de prevención: Investigar previamente si la tecnología elegida permite desarrollar completamente la funcionalidad deseada.

Plan de contingencia: En caso de que con la tecnología elegida no resulte tan útil como preveía, buscar una alternativa y paralelamente pensar qué otra funcionalidad se le podría dar a la tecnología descartada.

- Incompatibilidad de tecnologías aplicadas:

Suele ser un caso extraño cuando ya se ha trabajado previamente con las tecnologías pensadas, pero, sería un inconveniente que, llegado un punto de la implementación donde sea necesario juntar dos funcionalidades implementadas de manera separada, estas resulten incompatibles o que no sean fácilmente integrables.

Probabilidad: Muy baja

Efecto: Insignificante

Plan de prevención: Informarse previamente de la compatibilidad de las tecnologías y buscar ejemplos donde haya una integración similar.

Plan de contingencia: Dado el caso de no poder de ninguna manera compatibilizar las tecnologías, buscar una alternativa a aquella que sea menos compatible generalmente con las demás.

- Pérdida o corrupción de proyecto (aplicación):

  Durante la implementación de la aplicación se puede dar el caso de que los archivos de una funcionalidad concreta o del proyecto conjunto dejen de funcionar por algún error o se eliminen por equivocación.

  Probabilidad: Moderada

  Efecto: Serio

  Plan de prevención: Realizar constantes commits en el repositorio y utilizar GithHub para tener control total de las versiones. Además, implementar en la medida de lo posible las funcionalidades paralelamente al proyecto o al menos hacerlo con una copia previa de este y añadirla una vez esté completamente funcional sin errores.

  Plan de contingencia: Si se corrompe el proyecto en el que se está trabajando o deja de funcionar correctamente tras añadir una funcionalidad, volver a aquella versión donde funcionaba con normalidad.

- Caída de servidor de pre-producción o producción:

  Una vez implementada la aplicación completamente y estar lista para la producción es posible que los servidores donde esté alojada no estén en servicio o que esté alterado. Sería imposible entonces realizar una demostración real y legítima de la aplicación.

  Probabilidad: Muy baja

  Efecto: Moderado

  Plan de prevención: Informarse bien de la calidad de los servicios de hosting elegidos para disminuir el riesgo. Tener un servidor alternativo de emergencia (Preproducción y Producción).

  Plan de contingencia: Si resulta imposible acceder hay dos posibilidades: tener un video de demostración preparado o tener una simulación local en el ordenador propio.

- Ataque de terceros a la aplicación:

  Una vez la aplicación está lanzada en producción, esta se expone a los riesgos que tiene cualquier aplicación web. Se puede sufrir un ataque que malintencionadamente trate de corromper la seguridad de la plataforma y esto suponga pérdida de datos e incluso exposición de datos.

  Probabilidad: Baja

  Efecto: Serio

  Plan de prevención: Reforzar la seguridad del servidor, trabajar con el protocolo https, proteger la API con autentificación basada en token y recoger solo los datos sensibles estrictamente necesarios. Además, realizar copias de seguridad de la base de datos por si se ve afectada.

  Plan de contingencia: Si se vulnera la seguridad de la plataforma inhabilitar el servicio de manera urgente para evitar que vuelva a suceder antes de corregir el error o vulnerabilidad. Estudiar la manera en que se ha realizado el ataque y corregir las debilidades.

3. Planificación

A continuación, se especifican las fechas límite establecidas de cara a presentar el TFG en la convocatoria de junio C3. Dado que el día que se realiza esta planificación (10/03/2021) es ya una fecha bastante avanzada, los tiempos de realización de los apartados de la memoria serán muy justos, por lo que resultará una planificación exigente y con poco margen de retraso. Cada fecha marca la fecha límite de la entrega de este documento para la revisión del tutor, a fin de obtener su feedback.

En la tabla 3.1 se especifican estas fechas con los apartados de la memoria a revisar en cada una de ellas. Esta es la planificación a seguir en cuanto a la memoria, respecto la implementación, por tener también un gran número de tareas, se especificará más adelante.

Como herramienta de apoyo para esta planificación he empleado Trello. Con esta herramienta se puede crear un tablero con listas de tareas y el estado de estas. Este es el estado inicial del tablero de Trello, con las tareas respectivas a la memoria, con una tarea por cada apartado del documento y en los casos de haber sub-apartados, estos están especificados con un "checklist" dentro de la tarea.

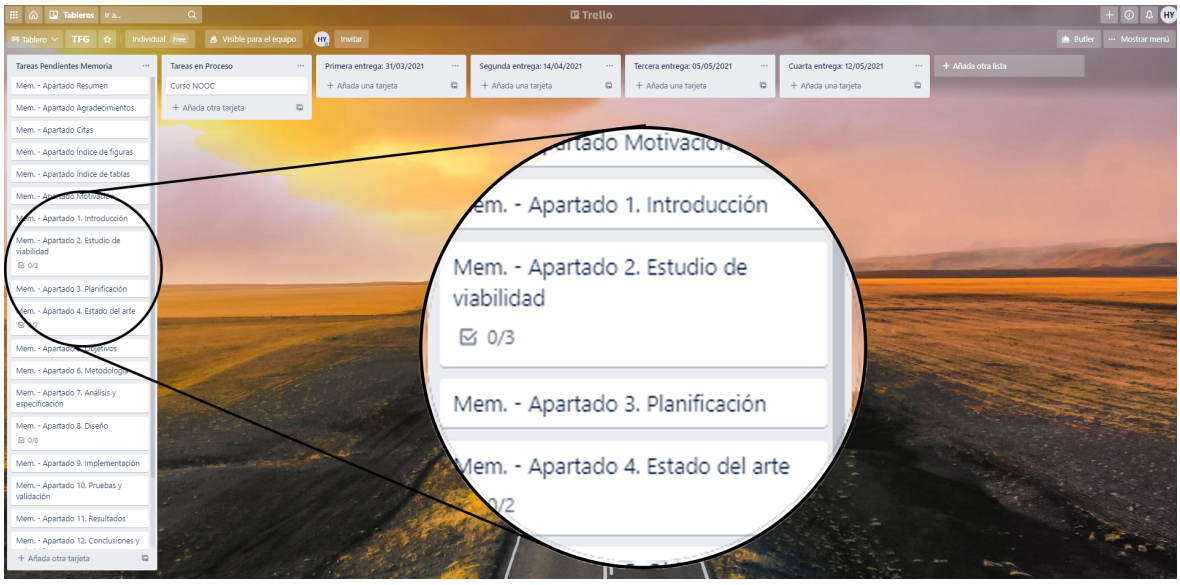


Figura 3.1: Estado del tablero de Trello a fecha 10/03/2021
Fuente propia

Contenidos	Tiempo total	Fecha límite fin
Motivación, justificación y objetivo general Estudio de viabilidad Introducción Estado del arte Planificación	3 semanas	31/03/2021
Objetivos Metodología Análisis y especificación Presupuesto y estimaciones Diseño	2 semanas	14/04/2021
Implementación	3 semanas	05/05/2021
Pruebas y validación Resultados Conclusiones y trabajo futuro Referencias, bibliografías y apéndices Agradecimientos, citas, índices	1 semanas	12/05/2021

Tabla 3.1: Planificación temporal TFG

En la primera columna observamos todas las tareas. En la segunda las que se están realizando. Y la tercera, cuarta, quinta y sexta, corresponden a las tareas que se han realizado para la entrega respectiva a la fecha que tienen en el título.

4. Estado del arte

4.1. Alternativas actuales

El estudio del mercado resulta muy útil para analizar la competencia a la que estará expuesta la aplicación desarrollada desde el momento en que se pone en producción. Si bien encontrar un número considerable de alternativas puede suponer una dura competencia inicial o una difícil inserción en el mercado, esta variedad se puede aprovechar para tener referencias tanto positivas a seguir, como negativas para mejorarlas. De esta manera, los aspectos positivos observados nos garantizarán en cierta manera no errar en lo más básico y familiar para el usuario y los aspectos negativos serán los que la propuesta mejore y por tanto haga decantarse a los usuarios. A esta mejora, obviamente, habrá que añadir una propuesta de valor única que no incluyan las demás.

A continuación, se describen brevemente aplicaciones que tienen funcionalidades similares. Estas alternativas han sido encontradas en Play Store tras hacer una búsqueda genérica con los términos "Compartir coche". Teniendo en cuenta que los resultados que ofrece esta tienda de aplicaciones están ordenados por popularidad, las alternativas analizadas también seguirán el mismo orden. Como criterios son considerados el número de descargas y la valoración de los usuarios.

Partiendo de la base de que todas ellas tienen una funcionalidad común con dos tipos de usuarios: el conductor que oferta plaza en su coche y el pasajero que busca una plaza, este análisis se centrará en los aspectos diferenciales de cada una. Por cada una de ellas se incluyen algunas capturas de las interfaces con el objeto de reunir similitudes y tomarlas como referencia.

Para realizar el análisis, además de leer las descripciones de cada una de ellas a fin conocer lo que ofrecen, han sido descargadas cada una de ellas y creada una cuenta de usuario para valorar la primera impresión que tendría un usuario al descargarla y empezar a utilizarla. Esta es toda la experiencia de uso que se ha podido reunir, por lo que no se puede valorar con precisión exacta si cumplen con lo que ofrecen ya que resulta inviable realizar un viaje en cada una de ellas solo para comprobar su funcionamiento. Se da por hecho que con la valoración de cada aplicación en Play Store es posible conocer la satisfacción de los usuarios.

4.1.1. BlaBlaCar

Sin ninguna duda es la aplicación más popular en este ámbito. Con más de 50 millones de descargas (contabilizadas en Play Store) y una puntuación de 4,5 estrellas, es evidente que se trata de uno de los referentes a seguir. Sin muchas más funcionalidades que la descrita, podemos destacar la posibilidad de crear alertas en caso de no encontrar un viaje para cierta fecha para que cuando se publique uno similar ser notificado. Además, contiene un apartado de mensajes que posibilita la comunicación entre pasajero y conductor para realizar consultas o matizar detalles del viaje. Sin embargo, en esta aplicación no se ofrece la posibilidad de tener un horario o calendario con el que poder definir un viaje rutinario, es decir, en todo momento se trata de buscar u ofrecer viajes eventuales.

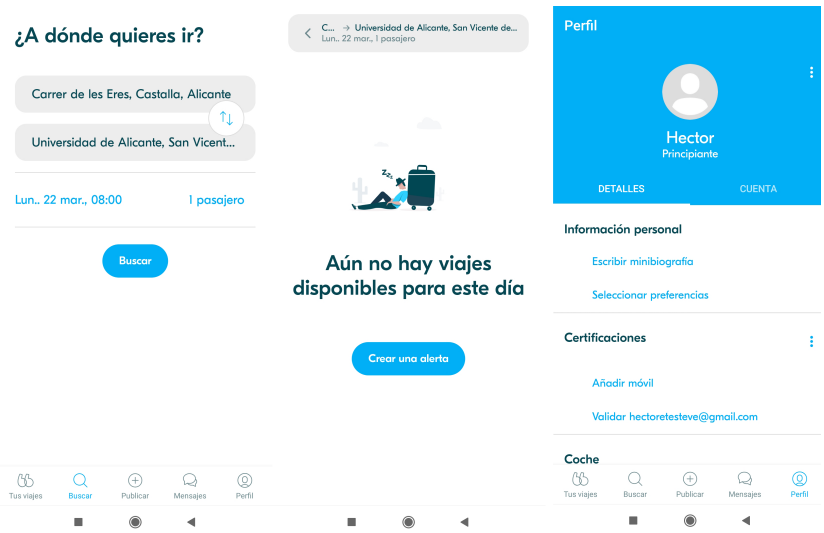


Figura 4.1: Capturas de pantalla de BlaBlacar
Fuente: Aplicación BlaBlacar

4.1.2. Amovens

El segundo resultado de la búsqueda y por tanto la segunda alternativa más popular. El número de descargas, sin embargo, es considerablemente inferior a la primera: 1 por cada 100, es decir, aproximadamente 500.000. La puntación también es alta: 4,2 lo que significa que cumple su función satisfactoriamente. No obstante, tiene un aspecto diferencial de esta respecto a BlaBlaCar y las demás aplicaciones que nos hace pensar que este sea el verdadero atractivo para los usuarios. Se trata del alquiler del vehículo propio y por el considerable número de usuarios parece un campo interesante pero que, dadas las características de la aplicación a desarrollar, no resulta muy aplicable. Además, Amovens incluye en el perfil de usuario un apartado de Pago, ofreciendo la posibilidad de realizar pagos por medio de la aplicación con tarjeta de crédito/débito e incluso una gestión de saldo para utilizar en la aplicación. A esto se añade un sistema de puntos canjeables para realizar alquileres gratuitamente.

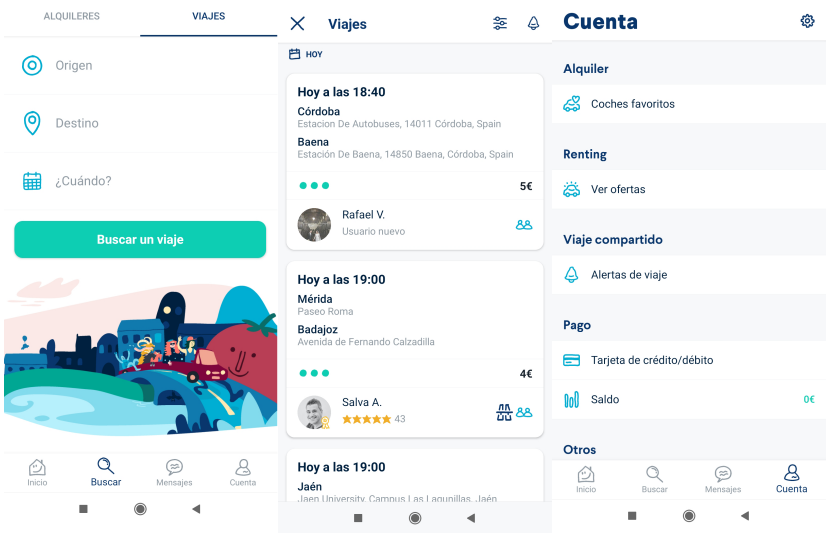


Figura 4.2: Capturas de pantalla de Amovens
Fuente: Aplicación Amovens

4.1.3. Journify

Journify es la primera aplicación que añade al hecho de compartir coche un factor extra a tener en cuenta por ser similar al buscado. El número de descargas es de nuevo considerablemente inferior a su antecesor, y curiosamente también con relación 1 a 100, con aproximadamente 5.000. El factor diferencial es que incluye la posibilidad de incluir una rutina. Esto es, si el viaje o viajes ofrecidos van a ser repetidos periódicamente, esta aplicación permite introducir tus viajes diarios durante la semana indicando las horas de ida y de vuelta. Por esta razón, esta alternativa es digna de ser considerada como ejemplo.

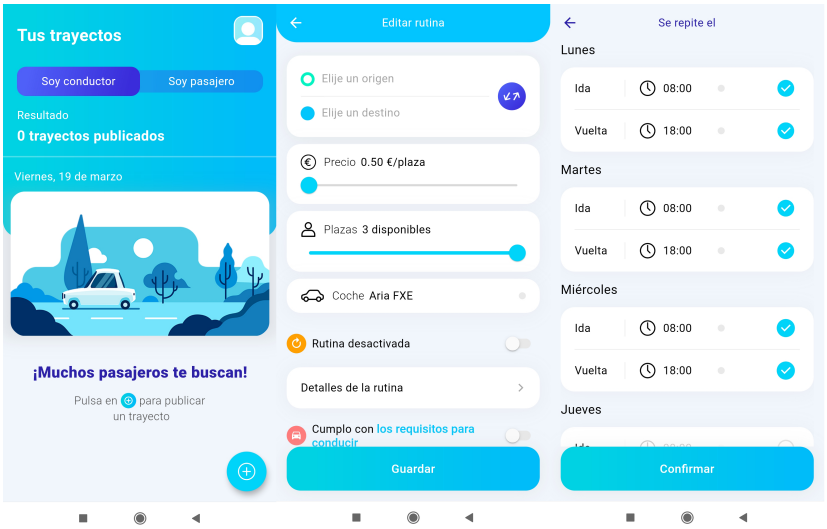


Figura 4.3: Capturas de pantalla de Journify
Fuente: Aplicación Journify

4.1.4. Amicoche

Esta aplicación tiene un comportamiento muy parecido a BlaBlaCar. En ella no se observa ninguna funcionalidad que la diferencie de las 3 anteriores. El número de descargas es de más de 10.000 y la puntuación de 4,1 estrellas. Como aspecto destacable y en el cual hacen hincapié en su descripción es el hecho de no cobrar comisiones en los pagos mediante la aplicación. Quizás por esta razón es la primera vista hasta el momento con publicidad. En definitiva, cumple con la función de ofrecer y encontrar viajes, pero sin tener ninguna funcionalidad diferencial que aporte valor.

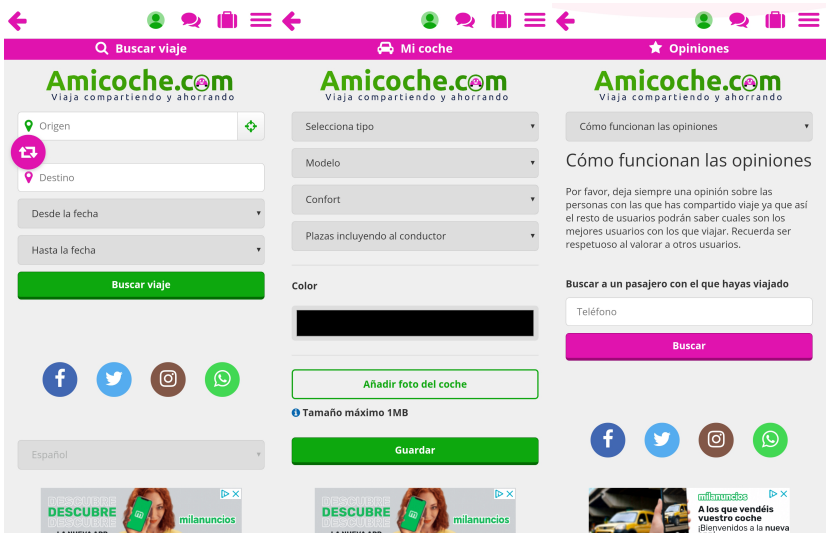


Figura 4.4: Capturas de pantalla de Amicoche
Fuente: Aplicación Amicoche

4.1.5. HoopCarpool

Muy semejante a la alternativa anterior, pero con una funcionalidad añadida aparentemente interesante. Número de descargas bastante inferior a las grandes alternativas: más de 1.000 y puntuación de 4,0 estrellas. Esta aplicación, según su descripción, ofrece la posibilidad de "calcular el precio de los viajes automáticamente, asegurando que el precio sea justo para ambas partes". Y relativo a también a los pagos en la aplicación, esta, al igual que Amovens, también tiene un punto diferencial en el modelo de negocio. En este caso ofrecen gratuitamente uno de cada cinco viajes compartidos que realices.

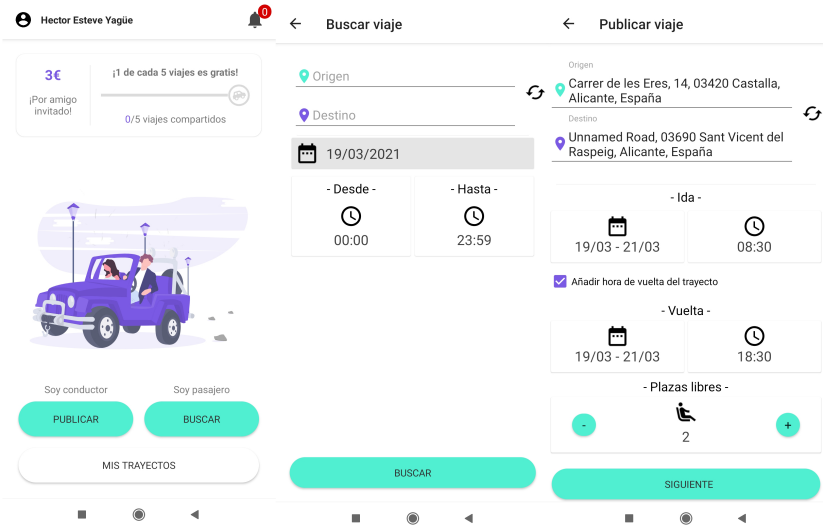


Figura 4.5: Capturas de pantalla de HoopCarpool
Fuente: Aplicación HoopCarpool

4.1.6. RACC Hop

Como última opción de la Play Store analizada es de todas estas la menos popular. Apenas 500 descargas y sin valoración visible en la ficha de la aplicación. De ella se distingue un elemento único respecto a las demás. En su descripción hacen mención a la posibilidad que ofrecen de realizar un convenio con universidades o empresas para crear redes privadas que gestionen viajes compartidos entre personas de una misma organización. Este aspecto sería realmente interesante de incluir en el modelo de negocio de cara al futuro.

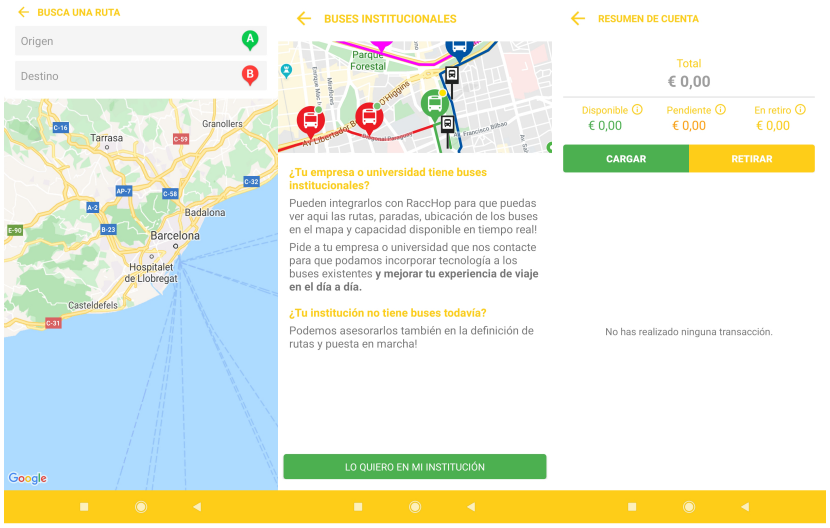


Figura 4.6: Capturas de pantalla de RACC Hop
Fuente: Aplicación RACC Hop

4.1.7. Conclusiones Aplicaciones Play Store

Como conclusión tras haber tenido contacto con las aplicaciones he observado que en todas y cada una de ellas es muy sencillo buscar viajes al igual que ofrecerlos. La interacción es común a todas, introducir origen y destino y fecha del viaje. En algunas de ellas se ofrece la posibilidad de marcar en un mapa tanto el origen como el destino, lo cual es visualmente atractivo, pero no lo considero estrictamente necesario y cómodo para un uso habitual. Otro punto que he visto bastante común es el sistema de alertas, el cual te avisa en caso de que se oferte un viaje que coincida con criterios que anteriormente no habías encontrado, lo cual es un aspecto interesante. También presente en todas las aplicaciones, por resultar útil, he visto un sistema de mensajería para poner en contacto al conductor y el pasajero. Asimismo, opinar sobre los usuarios con los que se comparte viaje es posible en casi todas ellas, con el fin de mostrar en el perfil del usuario una valoración, un aspecto muy a tener en cuenta en el momento de elegir el compañero de viaje. Y por último, en cuanto a los pagos he observado que generalmente en la aplicación se dispone de un saldo que fluctúa según ofertes tu coche o viajes con otro.

4.1.8. Autocolega UA

Además de las alternativas explicadas anteriormente, es a considerar una campaña impulsada por la propia Universidad de Alicante la cual tiene el propósito de fomentar el hecho de compartir coche. Se trata de una plataforma que funciona a modo de tablón de anuncios donde los interesados pueden publicar sus horario y trayecto para encontrar a gente con la que desplazarse y ahorrar gastos. Sin embargo, a pesar de ser una iniciativa muy interesante

por parte de la universidad, no ha llegado a tener un éxito considerable, pues el número de publicaciones que se pueden encontrar es insignificante. Una prueba de ello es que en la misma página del listado encontramos publicaciones con fechas de entre el año 2012 y 2020.

Así pues, no podemos considerar esta alternativa como un rival imposible de batir ya que no ha tenido mucha aceptación entre los universitarios. No obstante, resulta muy significante que la universidad esté interesada en fomentar el uso compartido del coche para desplazarse al campus. Esto se puede entender como una posible vía de negocio futura, pudiendo llegar a convenios u otro tipo de acuerdo para extender el uso de la aplicación.

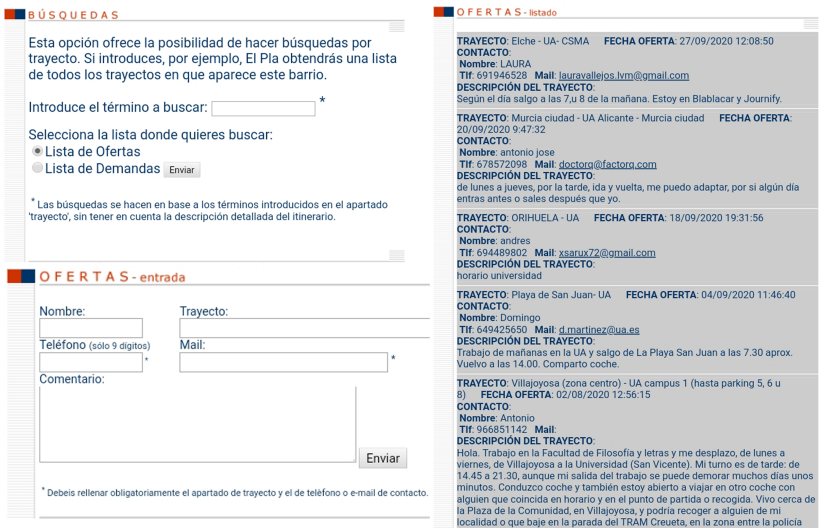


Figura 4.7: Capturas de pantalla de Autocolega UA
Fuente: `http://aplicacionesua.cpd.ua.es/autocolega/index2.htm`

4.2. Tecnologías para el desarrollo

Una vez vistas las alternativas actuales y valorados los puntos fuertes que permitirán a la aplicación entrar en el mercado, es momento de considerar qué tecnologías serán aplicadas para el desarrollo.

4.2.1. Smartphone como dispositivo objetivo

El primer punto a valorar es con qué tipo de dispositivo se pretende que se realice mayoritariamente el uso de la aplicación. Con el propósito de permitir el acceso a la aplicación desde cualquier dispositivo, esta será desarrollada para navegador. A pesar de ello, las funcionalidades ofrecidas y la interacción del usuario serán determinantes para decidir cuál es dispositivo objetivo. De esta manera, el diseño se realizará en primer lugar acorde con las

características de este dispositivo. Las dos opciones más extendidas y por tanto a considerar son el ordenador y el teléfono móvil, más concretamente smartphone.

Desplazarse a la universidad es un hábito diario y, por tanto, requiere una planificación constante. La interacción del usuario con la aplicación será frecuente y por ello es fundamental que sea rápida para que no suponga una excesiva pérdida de tiempo. Buscaremos aquel dispositivo que más accesibilidad ofrezca al usuario y entre las dos mencionadas, la más idónea en este aspecto es el Smartphone.

A día de hoy es complicado encontrar una persona que no haga uso diario de su teléfono móvil. Y más extraño es en el caso de los universitarios, en su gran mayoría jóvenes y más que familiarizados con las nuevas tecnologías. El smartphone ofrece al usuario multitud de servicios y por esta razón su uso ha generado cierta dependencia. La aplicación a desarrollar sería un servicio más, lo que no supondría un cambio radical en sus costumbres.

Ha sido tal el impacto de los smartphones que para el acceso convencional a internet están actualmente incluso superando al ordenador. Prueba de ello es la tendencia que muestra el gráfico de la figura 4.8. En él, se puede apreciar cómo en enero de 2010 el uso del navegador en móvil era inapreciable y a día de hoy supera al ordenador.

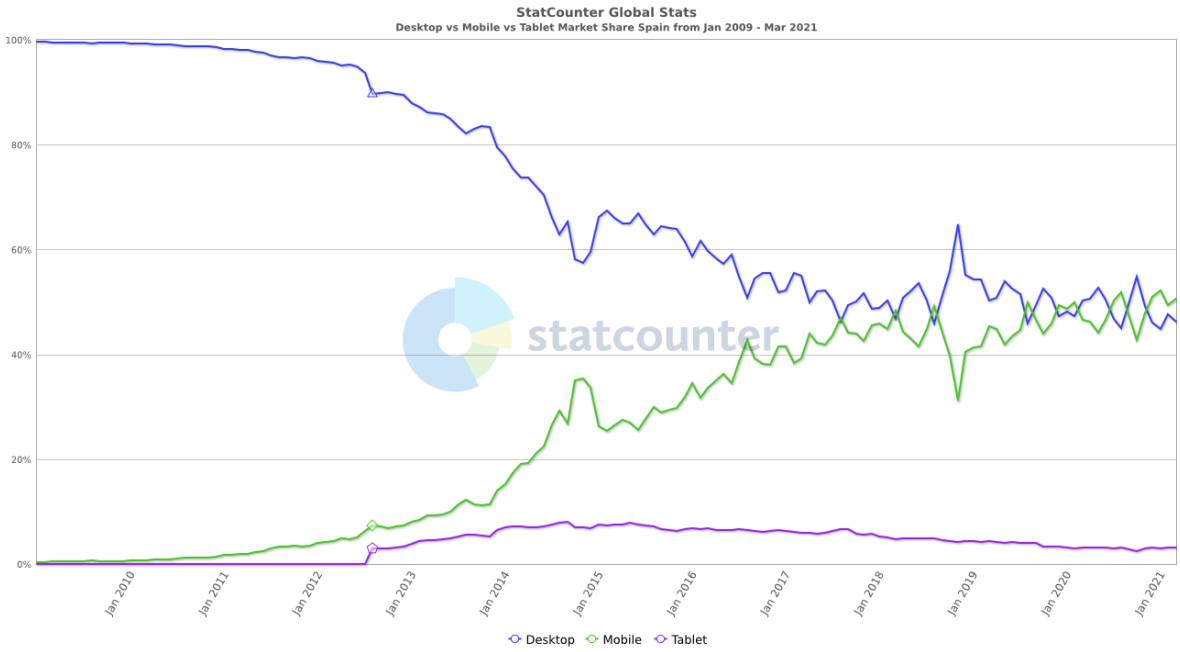


Figura 4.8: Comparación uso de navegador en escritorio, móvil y tablet
Fuente: `https://es.statcounter.com/`

Un ejemplo de servicio del smartphone que sería trasladable al que ofrecería la aplicación es la agenda o calendario. Resulta muy cómodo tener acceso desde el teléfono móvil a la

propia agenda y más interesante resulta si ésta te notifica cuando se acerca un evento. De forma similar la aplicación permitiría al usuario saber cómo y con quien se ha de desplazar a la universidad diariamente.

El concepto de diseñar primeramente una versión optimizada de la aplicación web para dispositivos móviles se conoce como Mobile First (RyteWiki, 2021). Más allá de las ventajas expuestas anteriormente que están relacionadas con la funcionalidad, hay aspectos técnicos que se benefician de esta práctica. Uno de ellos es el posicionamiento en los buscadores ya que estos valoran gratamente que el contenido esté adaptado para móvil. Además, se prioriza el hecho de limitar el contenido mostrado a tan solo el estrictamente necesario. Esto mejora la experiencia del usuario por dos razones: menores tiempos de carga y mejor usabilidad de la aplicación.

Por estas razones, tenemos en el Smartphone el dispositivo idóneo para que el usuario interactúe cómodamente con la aplicación. Además, el hecho de desarrollarla para navegador permitiría que esta fuera accesible también desde ordenador, siendo este un punto a favor.

4.2.2. PWA

Si bien es cierto que en el anterior apartado se ha considerado que el desarrollo para navegador es el ideal para permitir un acceso multiplataforma, el hecho de enfocar el diseño y funcionalidades para un uso habitual mediante smartphone puede resultar algo contradictorio. Generalmente, el desarrollo de aplicaciones para móviles se realiza de forma nativa en contraposición al uso de navegador. Sin embargo, con la irrupción de las PWA (Progressive Web App) este planteamiento cambia.

Una Aplicación Web Progresiva es una aplicación que se ejecutan desde el navegador y por tanto no depende de ningún sistema operativo. Este es su principal punto favorable. Además, mediante el uso de Sevice Workers son capaces de comportarse como aplicaciones nativas (Ramírez, 2018). Este comportamiento es tan similar que incluso pueden llegar a instalarse en el dispositivo (ordenador o smartphone) consiguiendo una apariencia idéntica a cualquier aplicación nativa descargada desde una tienda.

Resaltando esta última característica, en las PWA conseguimos que el usuario, a pesar de no ser una aplicación nativa, sea capaz de familiarizarse rápidamente y tenga un acceso tan rápido como cualquier otra aplicación.

Además, son desarrolladas como las webs convencionales y sus bases son los estándares web abiertos: HTML, CSS y JavaScript, lo que supone una gran ventaja respecto al desarrollo específico de aplicaciones nativas para los diferentes sistemas operativos. No es, por tanto, necesario aprender uno o varios lenguajes o framework concretos según el dispositivo donde se vaya a instalar.

En cuanto a funcionamiento, son dos las características que permiten un comportamien-

to similar al de una aplicación nativa: Service Workers y Applicaction Shell Architecture (Carmona, 2019).

Por una parte, los Service Workers son una secuencia de comandos que se ejecutan en segundo plano independientemente de la interacción del usuario. Esto nos permite funcionalidades muy interesantes y propias de las aplicaciones nativas como son las notificaciones push y la sincronización en segundo plano (Gaunt, 2020).La instalación de una PWA consiste en registrar un Service Worker el cual se inicia al arrancar la aplicación y que manejará las peticiones del dominio. El caché de este servicio se guarda en el navegador, por lo que el contenido de la aplicación podría ser visible incluso sin conexión.

La arquitectura Application Shell es la que permitirá una carga instantánea de las interfaces de la aplicación. Se trata del código HTML, CSS y JavaScript mínimo necesario para dichas interfaces, que es guardado en el caché y por tanto no se recarga cada vez que se visita una interfaz. Aplicar este tipo de estructura resulta beneficioso ya que se consigue un alto rendimiento con interacciones de tipo nativo por su rapidez y con un peso de datos minimizado (Osmani, 2019).

4.2.2.1. Ventajas de las PWA

Llegados a este punto, es conveniente investigar sobre las ventajas e inconvenientes de las PWA respecto a las aplicaciones nativas y si pueden ser una alternativa para desarrollar la aplicación objetivo.

En cuanto a ventajas, la primera de ellas es la similitud con las aplicaciones clásicas. Al ser posible incluso la instalación de la misma, para el usuario no supondrá ningún esfuerzo asimilar su funcionamiento. La aplicación será accesible desde la pantalla de inicio del smartphone y una vez dentro de ella, la interacción es totalmente igual a las nativas.

El bajo coste que supone tanto su desarrollo como su mantenimiento también es una ventaja. Al estar basadas en los lenguajes de programación web estándar, son fácilmente programables incluso pudiendo añadir un framework, tal y como se haría en una web convencional. Además, una vez puesta en producción las actualizaciones no requieren descargas ya que la web es procesada por el navegador y por tanto siempre se accederá a la última versión.

Además, los recursos que requieren son menores que las aplicaciones nativas. A pesar de estar instaladas, estas ocupan menos espacio ya que el código instalado no será el propio de la aplicación sino el *Service Worker*. El código de la aplicación, como se ha descrito anteriormente, será almacenado en el caché del navegador y por ello será similar al de una web convencional.

Otro aspecto relevante sobre las PWA con el que aventaja a las aplicaciones nativas es la visibilidad. La aplicación estará publicada en un sitio web razón por la que será indexada en los motores de búsqueda (*Introduction to progressive web apps - Progressive web apps (PWAs)*

| *MDN*, 2021). Esto supone que sea posicionada de forma natural a medida que se haga uso de ella. No será necesario una web auxiliar mediante la cual se haga publicidad y se visibilice en los motores de búsqueda sino que será la propia aplicación la que aparezca como resultado cuando se realice una búsqueda.

También por estar alojada en un sitio web, será accesible desde cualquier dispositivo que tenga navegador. Esto supone la accesibilidad desde los smartphones, que son el dispositivo objetivo y los ordenadores como alternativa. Por esta característica, incluso se podrá probar antes de ser instalada, dejando en manos del usuario la elección sobre si instalarla o no. Asimismo, la disponibilidad de la aplicación no estará sujeta a ninguna tienda de aplicaciones, pudiendo ser instalada en la pantalla de inicio directamente desde la web.

Para finalizar con las ventajas, por lo que respecta a la seguridad, las PWA pueden resultar incluso más seguras que algunas aplicaciones nativas. Esto es debido a que no tienen acceso a partes del sistema que sí serían accesibles desde la aplicación nativa (Carmona, 2019). Además las conexiones están realizadas bajo el protocolo HTTPS, lo que garantiza que el contenido enviado no haya sido manipulado.

4.2.2.2. Inconvenientes de las PWA

En cuanto a los inconvenientes, el hecho de no ser una aplicación nativa que se instale desde una tienda de aplicaciones, puede resultar algo chocante respecto a la manera más asimilada por los usuarios de instalar una aplicación. Sin embargo, esta diferencia no supone mayor complejidad en la instalación, por lo que el usuario podrá realizarla sin problemas.

Este inconveniente está directamente relacionado con el desconocimiento general sobre esta tecnología. Los usuarios, más familiarizados con las clásicas aplicaciones nativas, pueden ser reacios a un cambio, lo que sería un obstáculo inicial (López, 2020). Para minimizar este posible rechazo, sería conveniente una breve y clara explicación en la propia web con la el usuario pudiera ser sabedor de que la PWA tendrá un funcionamiento idéntico al de las aplicaciones tradicionales.

Por lo que respecta a las funcionalidades, las PWA son más limitadas por falta de acceso a ciertas funciones del dispositivo donde están instaladas. Al estar gestionadas por el navegador, funciones como los pagos por NFC o acceso a contactos, entre otras, que son propias de aplicaciones nativas no estarían disponibles. A pesar de ello, con la evolución de las tecnologías, este tipo de limitaciones cada vez son menores y es de entender que en un futuro próximo desaparezcan.

Y por último, un aspecto más técnico sobre los inconvenientes resulta ser el mayor consumo de batería. Esto se debe a que los códigos que se emplean son web y no están optimizados como las aplicaciones dedicadas a sistemas operativos concretos. Este es un inconveniente destacable pero se puede minimizar si se consigue que el uso de la aplicación sea rápido y cómodo para el usuario, disminuyendo así el uso.

4.2.2.3. Compatibilidad PWA con navegadores

Otra cuestión a tratar es la compatibilidad de las PWA con los navegadores. Como se ha detallado anteriormente, las PWA se ejecutan por medio de un navegador y por esta razón es necesario tener en cuenta la cantidad de navegadores que soportan esta tecnología.

Dos son las características esenciales para que la PWA se comporte de forma similar a una aplicación nativa. La primera es el Sevice Worker. Este es el requisito indispensable para que la aplicación pueda trabajar en segundo plano y por tanto estar pendiente de las peticiones del servidor. Por esta razón, resulta indispensable tener en cuenta cuales son los navegadores más utilizados por el público objetivo. En la figura 4.9 se ve claramente que Chrome y Safari cubren casi el 90% de la estadística y junto a Samsung Internet, el porcentajes es superior al 97%.

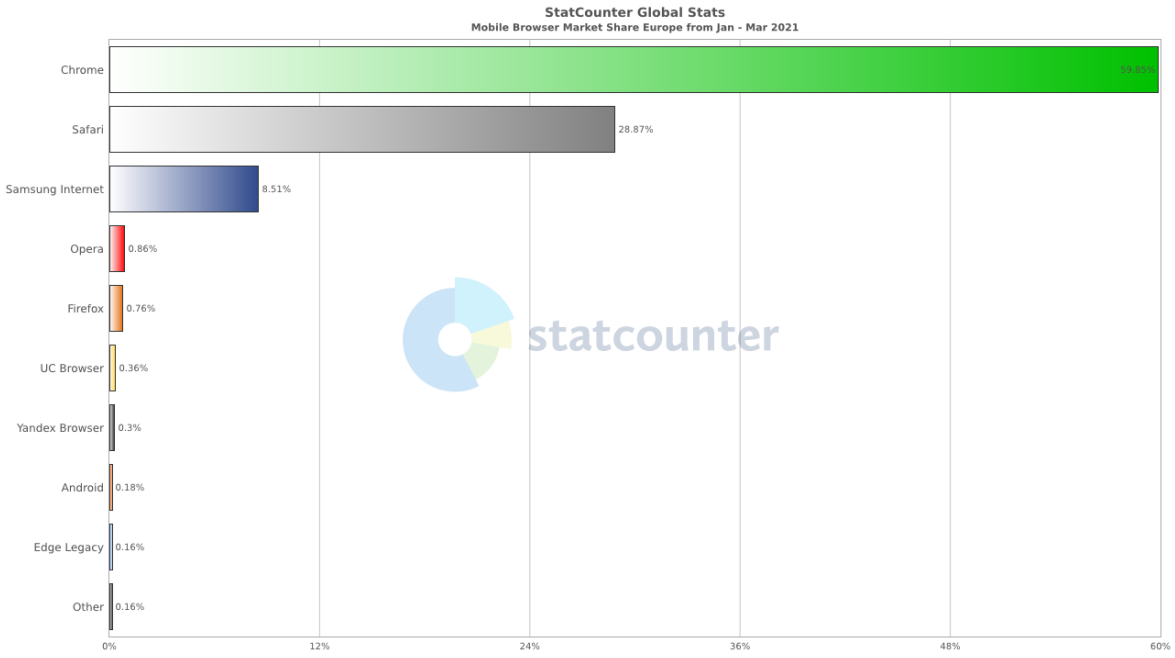


Figura 4.9: Uso de navegadores en móviles
Fuente: `https://es.statcounter.com/`

Una vez conocida esta estadística, cabe comprobar que la tecnología Service Worker es compatible con los tres navegadores, para poder de esta manera garantizar la posibilidad de uso de la PWA a casi la totalidad de los usuarios. La figura 4.10 presenta en verde aquella versión de los distintos navegadores en la que es compatible el uso de los Service Worker. A la vista está que los tres mencionados son compatibles.

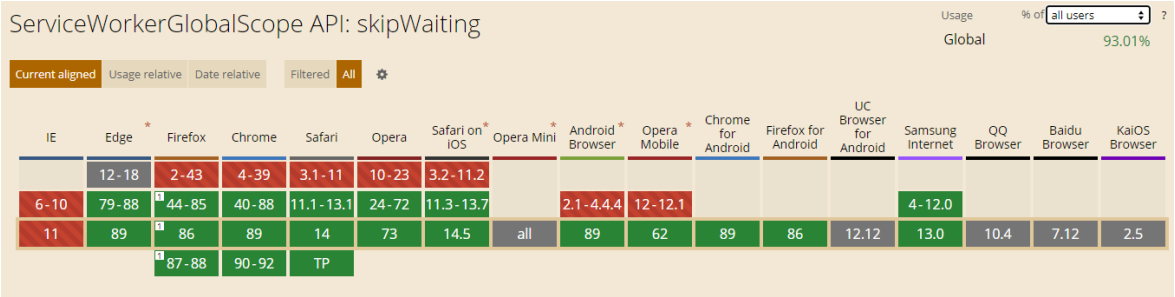


Figura 4.10: Compatibilidad Service Worker en navegadores
Fuente: `https://caniuse.com/`

La otra característica a tener en cuenta es la posibilidad de añadir la PWA a la pantalla de inicio desde la propia web. Esto es lo que dará la sensación al usuario de estar usando una aplicación tradicional. Tal y como vemos en la figura 4.11 la compatibilidad con los navegadores principales no es total. Los dispositivos con sistema operativo iOS que generalmente hacen uso de Safari todavía no soportan completamente esta característica. Sin embargo, a pesar de no tener esta funcionalidad, el propio navegador, desde sus ajustes, ofrece la posibilidad de añadir cualquier página web a la pantalla de inicio, que al fin y al cabo produce el mismo efecto. Para este nicho de usuarios, se debería especificar los pasos a seguir, que no son ni mucho menos complicados.

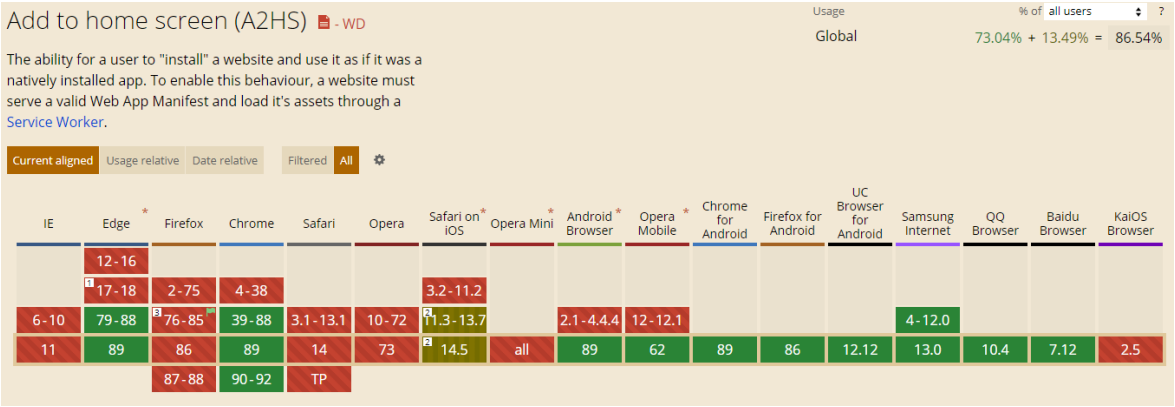


Figura 4.11: Compatibilidad "*Add to home screen*" en navegadores
Fuente: `https://caniuse.com/`

4.2.2.4. Conclusiones sobre las PWA

En definitiva, podemos considerar las PWA como una alternativa real frente a las aplicaciones nativas. Poniendo en una balanza sus virtudes y sus limitaciones expuestas anteriormente, la decisión se decanta a favor de los aspectos positivos. Y más concretamente para la aplicación a desarrollar donde las limitaciones comentadas no supondrán un obstáculo. El producto

final se podrá desarrollar con la libertad que ofrecen los lenguajes web estándar y además el comportamiento y aspecto final que tendrá de cara al usuario será muy similar a las aplicaciones nativas con las que ya está familiarizado. Además, será accesible desde cualquier dispositivo con navegador lo que permitirá que cualquier usuario pueda hacer uso de ella sin instalarla.

4.2.3. Desarrollo *Full-stack*: Node JS

Como entorno de ejecución para la aplicación se empleará Node JS. Este es un entorno basado en el motor JavaScript Chrome's V8 (*The Good and the Bad of Node.js Web App Development*, 2021) que nos permite gestionar de manera conjunta tanto el front-end de la aplicación como el back-end.

Node JS es una tecnología en auge y prueba de ello es que líderes tecnológicos como Uber, eBay, Paypal, Linkedin y Netflix hacen uso de ella para sus back-ends. Entre otras, las principales razones de su uso son el alto rendimiento que ofrece, la capacidad de implementar la lógica del negocio en el servidor y una codificación fácil y rápida (HackaBoss, 2020).

Para el desarrollo de la aplicación, son muchas las ventajas que ofrece Node JS frente a otros entornos de ejecución como PHP. Principalmente por haber desarrollado ya algún proyecto con Node JS, el hecho de tener JavaScript como lenguaje común y un acople total entre las dos partes permite un desarrollo rápido y sencillo que puede adaptarse por completo a las funcionalidades.

Por lo que respecta a la escalabilidad, Node JS presenta el paradigma E/S sin bloqueo, permitiendo de esta manera servir múltiples eventos simultáneos. Además, sigue el modelo basado en eventos, lo que garantiza una sincronización mayor entre el lado del servidor y el lado el cliente. Esto supone una gran ventaja de cara a futuro si la aplicación comienza a tener un volumen considerable de usuarios que entran simultáneamente.

Otro de los puntos a tener en cuenta es la gestión de los datos. Node JS utiliza JSON como formato de texto para el intercambio de datos tanto en la parte del servidor como en la parte del cliente, por lo que no se requiere ninguna transformación intermedia. De la misma manera, tiene librerías que permiten el uso de JSON para bases de datos relacionales y no relacionales. Esto permite tener más de una base de datos, cada una de un tipo según el propósito de los datos a almacenar pero a fin de cuentas los datos a tratar con formato JSON.

También relacionado con la gestión de los datos, Node JS permite implementar una API REST encargada de gestionar las peticiones del lado del cliente al servidor. Esta será desarrollada con el framework Express JS, sobre el que se profundizará mas adelante.

En definitiva, en el entorno Node JS tenemos la posibilidad de desarrollar todas las partes que componen la aplicación web: cliente, API REST y servidor. Además ofrece la suficiente

flexibilidad para implementarlas de manera sencilla y con todos los requisitos.

4.2.3.1. Express JS

Express JS es un paquete de Node JS que se define como un framework para web, especialmente para Node JS. Este framework nos permite gestionar las rutas de la aplicación tanto para la parte del cliente como para la parte del servidor. Por esta razón se puede desarrollar en él la propia interfaz y la API REST que gestiona las peticiones. Por estar implementado para Node JS, proporciona unas características de aplicación web básicas que potencian las capacidades de Node JS (*Express - Infraestructura de aplicaciones web Node.js*, 2021).

Además de gestionar las rutas, Express nos permite hacer uso de middlewares en la conexión cliente y servidor. Uno de ellos, fundamental para aplicaciones web con usuarios, es la autenticación y autorización. Este resulta necesario para verificar que la petición está realizada por un usuario con permisos.

Otro de los puntos fuertes de Express JS es el manejo de errores. De la misma manera que Node JS está basado en el modelo basado en eventos, Express JS se aprovecha de ello para tener control sobre los errores que se pueden producir durante su ejecución (*Manejo de errores de Express*, 2021). Estos, cuando son capturados son enviados a la parte del cliente en forma de error, por lo que se pueden manejar como tal e indicar al usuario final que el error producido ha sido interno. También resulta esto muy beneficioso para la depuración de la aplicación ya que se pueden capturar los errores específicos y en muchos casos la causa directa que los produce.

4.2.4. Hosting

El último aspecto a tratar sobre las tecnologías empleadas es el hosting. Una buena elección de este es fundamental para no encontrarnos sorpresas durante la puesta en marcha de la aplicación web. Puesto que el entorno de ejecución va a ser Node JS, es conveniente buscar aquellos servicios que ofrezcan compatibilidad con este.

4.2.4.1. Tipos de servicios

Son dos los tipos de servicios que se pueden contratar para alojar en ellos una aplicación: IaaS (Infraestructure-as-a-Service) y PaaS (Platform-as-a-Service). Estos se diferencian en el grado de mantenimiento que requieren, así como el control sobre los recursos instalados (Axarnet, 2021).

Por una parte, los IaaS ofrecen mayor libertad en cuanto a los recursos instalados pero

requieren más tiempo dedicado al mantenimiento y configuración del servidor. El proveedor ofrece máquinas virtuales donde instalar un sistema operativo a elegir y sobre este instalar el servidor. Además el desarrollador debe instalar todos los recursos necesarios para el funcionamiento de la aplicación, que concretamente son el entorno de ejecución Node JS y las bases de datos, en el caso de una aplicación básica. Sin embargo, la parte positiva es que todo puede estar gestionado en el mismo servidor, sin requerir servicios de terceros para alojar varias bases de datos, gestión personalizada de archivos (ftp) o incluso alojar otra aplicación o web complementaria.

Los hosting con PaaS, por otra parte, son sistemas donde el desarrollo de aplicaciones web resulta más sencillo. En contraposición a los IaaS, en este tipo de servicio solo es necesario ocuparse del mantenimiento de la propia aplicación sin tener en cuenta la configuración del servidor. El proveedor tiene su propia estructura de servidor en la que directamente se instala la aplicación, gestionando de manera "invisible" al desarrollador los procesos que se llevan a cabo internamente. Esto permite una instalación sencilla pero dependiente de los recursos que ofrece el proveedor. La base de datos, por ejemplo, puede estar limitada en cuanto al tipo y número. Esto supone buscar otros proveedores de servicios donde alojar más recursos como los mencionados anteriormente.

Puesto que el objetivo final es implementar una aplicación web, el servicio más recomendable es el PaaS. Sin embargo, por las limitaciones que puede ofrecer de cara a una producción mayor, resulta interesante plantear una solución con IaaS donde instalar de manera personalizada todos los recursos. Por esta razón en este apartado se explicarán dos entornos donde ejecutar la aplicación después de ser desarrollada de manera local, uno de pruebas servido con PaaS y otro de producción de cara a futuro con IaaS.

4.2.4.2. Entorno de Pre-producción

Este entorno seria el paso intermedio entre el desarrollo y la producción. En él se realizarán las pruebas de la aplicación de manera que se simulará el comportamiento de la aplicación antes de estar expuesto al uso por parte de los clientes. Tal y como se ha especificado antes, para este se empleará un PaaS, donde rápidamente se pondrá en funcionamiento la aplicación. Para mayor comodidad en cuanto a la instalación, se buscará una opción que ofrezca la posibilidad de instalar concretamente una aplicación Node JS. La opción elegida es Heroku.

4.2.4.3. Heroku

Heroku es uno de los PaaS más populares del mercado por ofrecer un plan gratuito y la posibilidad de elegir entre varios de los lenguajes de programación más extendidos como son Node JS, Ruby, Java, PHP y Python. A esto se le suma la integración con GitHub lo que permite una sencilla instalación de la aplicación desde un repositorio y la puesta en marcha desde una interfaz intuitiva con los los pasos a seguir bien marcados.

Por lo que respecta a Node JS, como se ha mencionado anteriormente, Heroku es totalmente compatible y la instalación resulta muy sencilla. Sin embargo, en cuanto a la base de datos, Heroku solamente ofrece la posibilidad de tener un tipo base de datos, SQL. Concretamente del tipo Ignite, el cual no está tan extendido como MySQL. La instalación de la base de datos de la aplicación se realiza mediante un "add-on" que es propiamente un servicio de base de datos (*ClearDB MySQL | Heroku Dev Center*, 2021). Si bien, tener una base de datos relacional sería posible en Heroku, en el caso de requerir una no relacional habría que buscar un servicio de terceros.

El principal inconveniente de Heroku son los precios. El coste del plan mínimo para producción supera a un hosting IaaS y además tiene limitación de peticiones por hora e incluso la posibilidad de que el servicio entre en modo de "suspensión" si durante un tiempo la aplicación no está en uso. Por esta razón Heroku solo se empleará para este entorno y el plan elegido es el gratuito que está orientado para realizar pruebas. Puesto que la configuración de la base de datos en Heroku requiere la instalación de "add-on" y esto, a su vez, la configuración de un método de pago. Para minimizar la configuración en Heroku, en cuanto a la base de datos se optará por emplear un servicio gratuito que solo requiere registro con correo electrónico. Se trata de la web `https://www.freemysqlhosting.net/`.

4.2.4.4. Entorno de Producción

El entorno de producción será aquel que tendrá la aplicación en funcionamiento y permitirá a los usuarios finales su uso. Por esta razón, este debe ser sólido y depender mínimamente de servicios de terceros. Derivar funciones a terceros significa en la mayoría de casos adaptarse a sus exigencias, lo cual puede suponer mayor tiempo configurando los recursos de cada servicio y más gasto económico si los planes gratuitos están limitados. Este es uno de los inconvenientes que conlleva Heroku. Su plan gratuito resulta limitado para la producción, por lo que sería necesario un plan de pago. Estos planes, además de ser caros, requieren adaptarse a la forma de desplegar aplicaciones propia de Heroku.

Así pues, si un entorno real de producción raramente va a ser posible con los planes gratuitos de los PaaS, nos veremos en la obligación de contratar a un servicio de pago. Tras buscar las diferentes alternativas donde desplegar una aplicación con Node JS, la conclusión es que los PaaS son excesivamente caros en comparación con los IaaS y más fastidioso incluso si, pagando, es necesario adaptarse a las tecnologías concretas con las que trabaja el PaaS contratado.

No es descabellado, por tanto, plantearse la opción de contratar un servicio IaaS donde alojar la aplicación y además poder instalar y configurar de manera personalizada cada una de las tecnologías a utilizar. Además esto nos permitirá instalar en este servidor complementos a la aplicación como podría ser una web corporativa para dar visibilidad a la aplicación.

4.2.4.5. Digital Ocean

De entre los múltiples proveedores de hosting el elegido es Digital Ocean. Las principales razones de la elección son su económico precio, la gran variedad de sistemas operativos que ofrece para instalar en la máquina virtual y la sencillez de la configuración del servidor. Además, valorando la escalabilidad, esta plataforma permite ampliar los recursos del servidor a medida que estos vayan quedando limitados por un posible crecimiento de la aplicación.

Como sistema operativo, la imagen elegida es Ubuntu 20.04 con el software para desarrollo de aplicaciones Docker 19.03.12. Esta es una plataforma de código abierto que permite la creación y uso de contenedores en los que se alojan diferentes aplicaciones o webs del servidor (*What is Docker?*, 2021). Gracias a esta estructura por contenedores, la aplicación podría ser desplegada en otra máquina independientemente de su configuración, ya que esta, la requerida por la aplicación, se ajusta a la del propio contenedor.

4.2.4.6. CapRover

Como última tecnología referente al hosting encontramos CapRover. Se trata de un administrador de servidores web y despliegue de aplicaciones y bases de datos que facilita la instalación y puesta en producción de estos. Para su instalación en el servidor requiere de Docker, por lo que está diseñado para trabajar de forma complementaria.

Como características principales, tras su instalación, CapRover habilita un subdominio en el servidor en el que se puede acceder a la parte de administración. En ella, de forma muy intuitiva se pueden desplegar tantas aplicaciones como sean necesarias, existiendo la posibilidad de elegir entre aplicaciones conocidas como Wordpress o Node JS, bases de datos MongoDB y MySQL e incluso aplicaciones personalizadas que no se ajusten a las predeterminadas.

El servidor sobre el que trabaja CapRover es Nginx. Este ligero servidor actúa como balanceador de carga y se puede personalizar según las exigencias, de manera que la seguridad puede ser reforzada. También respecto a la seguridad, CapRover permite la habilitación de manera muy sencilla del certificado SSL mediante Let's Encrypt.

En definitiva, podemos catalogar CapRover como un PaaS personalizable y a medida que ofrece los beneficios de este tipo de servidor y, además, no limita la configuración del servidor ya que está instalado en un servicio IaaS totalmente accesible.

5. Objetivos

Los objetivos definirán aquellas metas propuestas antes de iniciar el proyecto y servirán como referencia para conocer el éxito de este. El hecho de tener proyectados unos resultados futuros a conseguir resultará útil como motivación personal.

El objetivo principal de este proyecto es obtener una aplicación web completamente funcional y que su uso sustituya la función del grupo de WhatsApp mencionado anteriormente, lo que supondría un uso diario por parte de los usuarios. No obstante, este objetivo es muy genérico y resultaría bastante complejo determinar qué porcentaje de éxito se ha conseguido en caso de no conseguirlo por completo. Para ello se establecerán una serie de objetivos más concretos.

Es muy importante hacer una adecuada definición de estos subobjetivos por lo que se tendrá en cuenta el concepto "objetivos SMART" introducido por George T. Doran en la edición de noviembre de 1981 de la revista *Management Review*. Los acrónimos S.M.A.R.T significan *Specific, Measurable, Assignable, Realistic y Time-related*, lo que sería traducible como *Específico, Medible, Asignable, Realista y Relacionado con el tiempo*.

Fijar los objetivos teniendo en cuenta estos conceptos es verdaderamente interesante porque permite detenerse a pensar si cada uno de ellos cumple con los requisitos. De esta manera se evitan objetivos inalcanzables, generalistas y difícilmente precisos a la hora de determinar si han sido conseguidos o no.

Estos subobjetivos serán divididos en dos grupos. El primero de ellos está relacionado con los aspectos técnicos y la forma en que se programe la aplicación. Es decir, aquellos objetivos a tener en cuenta durante el desarrollo. Estos por tanto tienen una fecha de consecución común, que coincidiría con la entrega final del TFG. Estos objetivos son:

- Explotar al máximo las capacidades de las aplicaciones web y concretamente de las PWA para que, de cara al usuario, resulte inapreciable la diferencia entre *web app* y una aplicación nativa.

- Conseguir una PWA en la que el uso de las Notificaciones Push personalizadas imite el comportamiento de las notificaciones de las apps nativas.

- Lograr un bajo acoplamiento entre las tecnologías empleadas para que puedan ser cambiadas sin grandes inconvenientes.

- Dejar preparada la aplicación para poder expandir su uso más allá del grupo de estudiantes de la zona.

El segundo grupo de subobjetivos está relacionado con las metas de la aplicación y su éxito en el mercado, es decir, una vez la aplicación ya está en la etapa producción. Estos objetivos son:

- Conseguir que, de cara al inicio curso siguiente, 2021/2022, al menos la mitad de los integrantes del grupo de WhatsApp se hayan registrado en la aplicación.

- Lograr el uso cotidiano de un cuarto de los integrantes del grupo durante los 2 primeros meses del curso 2021/2022 y obtener una valoración positiva que sirva como estímulo para los demás integrantes.

- A finales del curso 2021/2022, haber sustituido por completo el uso del grupo actual de WhatsApp por el de la aplicación, estando registrados todos los integrantes y haciendo uso cotidiano.

- Mantener el buen funcionamiento de la aplicación con los usuarios del grupo de WhatsApp durante un periodo de 6 meses a modo de prueba para corroborar la solidez de la aplicación.

- Ampliar el alcance de la aplicación a estudiantes de pueblos de la zona de cara al curso 2022-2023.

- Llegar a un acuerdo con la universidad o con algún ayuntamiento para fomentar el uso de la aplicación como alternativa para desplazarse al campus.

6. Metodología

En este apartado se describen las diferentes metodologías que se combinarán durante el desarrollo del TFG. Es conveniente seguir los criterios estipulados por estas metodologías para que el tiempo empleado resulte productivo y se mantenga un orden concreto. Esto ayudará a saber en cualquier momento cual es el estado real del proyecto, conociendo exactamente qué falta y cuando debe estar listo.

En primer lugar, cabe distinguir los dos tipos de metodologías a aplicar: Waterfall y Agile. La primera ha sido la más utilizada tradicionalmente y consiste en un desarrollo lineal donde cada etapa no comienza hasta finalizar la anterior. La segunda en cambio es más nueva y el desarrollo es iterativo priorizando la rapidez. Las etapas pueden ser simultáneas siendo un desarrollo más propenso a cambios. En la figura 6.1 se puede observar de forma esquemática la diferencia entre ambas.

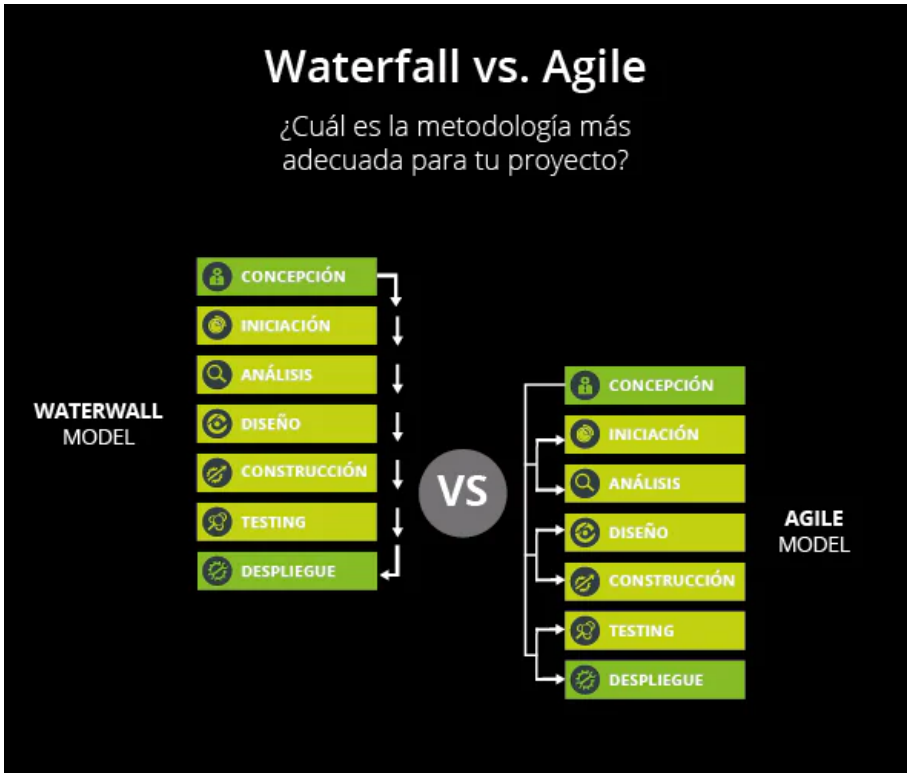


Figura 6.1: Comparación entre etapas de Waterfall y Agile
Fuente: `https://www2.deloitte.com/`

Para el desarrollo de este TFG se combinarán ambas metodologías. En primer lugar, los apartados previos a la implementación, como son análisis, especificación y diseño, se realizarán de forma secuencial, metodología Waterfall. De esta manera se evitará un error muy común en el desarrollo de software, como es comenzar a programar sin previamente tener claras las especificaciones. Este error supone una pérdida de tiempo porque muy difícilmente en el momento de empezar el proyecto se tengan claros todos los requisitos y por tanto se debe ir rectificando a medida que se programa. En cambio, dedicar suficiente tiempo a la especificación de requisitos y diseño ayuda a asentar y concretar las ideas poniendo en cuestión su validez antes de programar.

Una vez terminadas estas etapas, se iniciará la implementación. Para esta etapa, se aplicarán conceptos de las metodologías ágiles, más concretamente de la metodología Scrum, como son los *sprints*. En Scrum, los *sprints* son periodos de tiempo en los que se debe generar valor (*Las 5 ceremonias Scrum: claves para la gestión de procesos*, 2021). Es decir, al final de cada *sprint* se debe tener, por ejemplo, una funcionalidad completa y que se pueda demostrar su correcto funcionamiento.

Otro concepto relevante de las metodologías ágiles es la distribución del trabajo en tareas. Esta práctica consiste en dividir cada principal funcionalidad de la aplicación hasta conseguir en su mínima expresión los complementos necesarios para su implementación. Cada una de estas tareas se especificará en una tarjeta y estas a su vez se distribuirán en un tablero dividido en columnas. Las columnas determinan el estado de las tareas que están en ellas, y generalmente son 4: “Por hacer”, “En progreso”, “Pendientes” y “Terminadas”.

A fin de poder visualizar con mayor claridad el estado del sprint actual, a las 4 columnas mencionadas anteriormente se añadirá una quinta. Esta se situará entre “Por hacer” y “En progreso” y se llamará “Tareas del sprint”.

El proceso a seguir será el siguiente: En la primera columna se listarán todas y cada una de las tareas de la implementación. En la segunda, las que deben realizarse para el sprint actual. En la tercera, las que están en ese momento realizándose. En la cuarta, las que están pendientes de revisión para comprobar su correcto funcionamiento. En caso de no pasar correctamente la validación, estas volverán a la columna anterior para corregirlas. Y finalmente en la quinta, las que ya han sido comprobadas y su funcionamiento es el deseado.

Así pues, la herramienta empleada para visualizar la lista de tareas y su estado será Trello, al igual que se ha empleado en el apartado de Planificación. En este caso, cada tarea será una funcionalidad determinada en los requisitos especificados en el apartado de Análisis y especificación. Internamente, cada una de estas tareas puede tener una subdivisión, que permitirá precisar aún más la definición de las funcionalidades.

En la figura 6.2 podemos observar el estado inicial del tablero tras haber colocado las tareas en la primera columna. Del tablero podemos destacar los 4 grupos de tareas, distinguidos cada uno con un color. Estos cuatro grupos son los referentes a las tareas de Base de Datos, API, *Web App* y Despliegue en entornos, a los que corresponden los colores naranja, amarillo, verde

y rojo respectivamente.

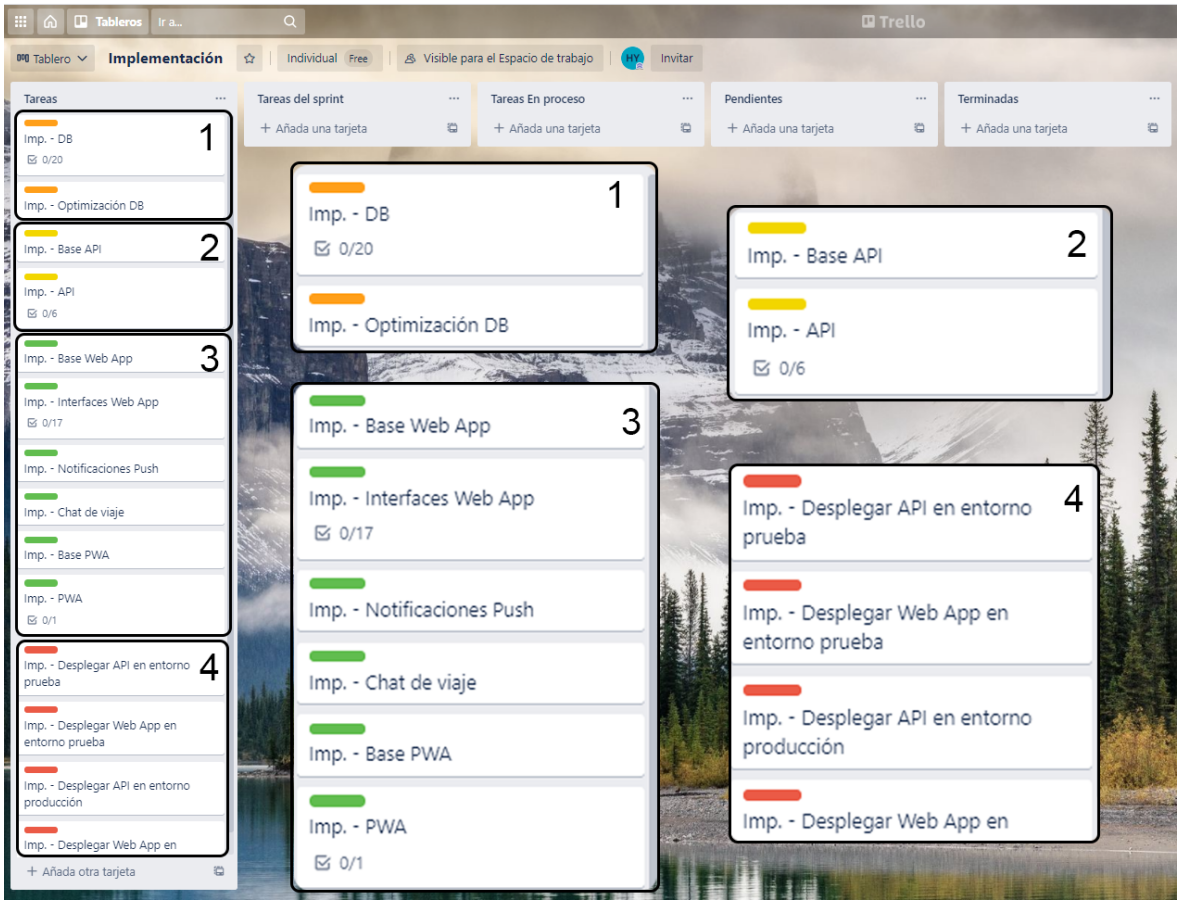


Figura 6.2: Estado inicial del tablero de Implementación
Fuente propia

7. Análisis y especificación

En el apartado de análisis y especificación de detallarán los tipos de usuario y los requisitos tanto funcionales como no funcionales de la aplicación. Para realizarlo, se seguirá el estándar de análisis y especificación IEEE 830 (*Especificación de Requisitos según el estándar de IEEE 830*, 2018).

En este documento no se desarrollarán todos los apartados que refleja dicho estándar. Sin embargo, tras aplicar esta plantilla quedarán definidos todos los aspectos necesarios para concebir la idea de la aplicación y posteriormente diseñarla.

7.1. Contexto

En primer lugar, con el objeto de contextualizar, se explicarán los conceptos clave y las funciones de la aplicación.

7.1.1. Definiciones del sistema

Estas definiciones serán sobre términos empleados durante la especificación. Se hará referencia a ellos con bastante frecuencia, por lo que es necesario matizar exactamente su significado en el ámbito de la aplicación.

- Viaje: Cada uno de los desplazamientos que se realizan desde una localidad hacia la universidad o viceversa. Según este criterio, un viaje será considerado como ida o como vuelta.

- Origen: En viajes de ida, localidad donde se inicia el viaje. En viajes de vuelta, universidad.

- Destino: En viajes de ida, universidad. En viajes de vuelta, localidad donde se termina el viaje.

- Pasos intermedios: En los viajes de ida o vuelta, localidades por las que el conductor se ofrece a pasar si algún algún pasajero es de distinta localidad.

- Conductor: Estudiante encargado de conducir el coche durante el viaje.
- Pasajero: Estudiante que acompaña al conductor durante el viaje.
- Hora de llegada: En los viajes de ida, la hora prevista de llegada a la universidad. Esta hora es independiente a la hora de salida que acuerden el conductor y los pasajeros.
- Hora de vuelta: En los viajes de vuelta, la hora propuesta inicialmente por el conductor para iniciar el viaje.
- Punto de encuentro: Lugar acordado entre conductor y pasajero para reunirse antes del viaje.
- Lista de conocidos: En el perfil de usuario, lista de usuarios con los que se tiene una mayor relación. Por ejemplo, usuarios cuya coincidencia en los viajes es recurrente.
- Horario: En el perfil de usuario, planificación diaria o semanal de horas de entrada y salida de la universidad.
- Calendario: En el perfil de usuario, planificación completa de los horarios del cuatrimestre.
- Viaje recurrente: Viaje que será repetido periódicamente y que será establecido por un conductor. Por ejemplo, un usuario que todos los martes ofrecerá plaza en su coche para llegar a la universidad a las 9:00.
- Valoración como conductor: Valoración que representará la opinión de los usuarios sobre un usuario cuando este actúa como conductor en un viaje.
- Valoración como pasajero: Valoración que representará la opinión de los usuarios sobre un usuario cuando este actúa como pasajero en un viaje.

7.1.2. Funciones del producto

Continuando con la descripción del contexto de la aplicación, se explicarán a grandes rasgos las funciones que esta ofrecerá a los usuarios:

- La aplicación permitirá a los usuarios ofrecer y buscar viajes para ir a la universidad o volver a su localidad.
- Los usuarios podrán planificar sus viajes a partir de su horario y/o calendario.
- La aplicación ofrecerá la posibilidad de usar recordatorios y avisos por medio de noti-

ficaciones.

- La aplicación realizará sugerencias sobre usuarios con horarios similares para planificar y establecer viajes recurrentes.

7.2. Tipos de usuario

En la tabla 7.1 están listados diferentes tipos de usuarios y una descripción sobre sus posibilidades en la aplicación.

Tipo	Descripción
Usuario	Usuario básico de la aplicación. Ante la participación en un viaje tendrá la posibilidad de actuar de dos maneras (roles) según sus necesidades: Conductor o Pasajero. Además, podrá introducir su calendario universitario para planificar sus viajes acorde con este.
Usuario (Rol Conductor)	El usuario que oferta un viaje desde una localidad hacia la universidad o viceversa. Podrá decidir la ruta del viaje (origen, destino y pasos intermedios), la hora de llegada o salida y los pasajeros que lo acompañen durante el viaje. Deberá acordar con los pasajeros la hora y el punto de encuentro.
Usuario (Rol Pasajero)	El usuario que solicita plaza en un viaje disponible. Podrá acordar con el Conductor detalles del viaje como la hora y el lugar de encuentro.
Administrador	Usuario con acceso a la parte administrativa de la aplicación para comprobar las estadísticas de esta, realizar el mantenimiento y resolver posibles incidencias (causadas por la aplicación) en los viajes.

Tabla 7.1: Tipos de usuario de la aplicación

7.3. Requisitos

A continuación están expuestos los requisitos de la aplicación. Estos se clasifican en dos categorías según tengan un efecto directo sobre las funcionalidades de la aplicación o sobre el comportamiento de esta.

Además de la descripción de cada uno de ellos, todos están identificados mediante un identificador que indica la categoría a la que pertenecen y el número de requisito. Además están asociados al tipo de usuario al que va asociada la funcionalidad. El cuarto aspecto que se detalla es el tipo. Estos tipos varían según la categoría.

7.3.1. Requisitos funcionales

Los requisitos funcionales serán aquellos que determinen los servicios que ofrezca la aplicación al usuario. En este caso, el tipo marcará la obligatoriedad y/o necesidad del requisito. Así pues, estos pueden ser básicos si son estrictamente necesarios para el funcionamiento mínimo o avanzados en caso de poder realizarse de cara a futuro. El prefijo del identificador es "RF".

Tabla 7.2: Requisitos funcionales de la aplicación

Identificador	RF1
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán registrarse e iniciar sesión en la aplicación creando su propia cuenta de usuario.

Identificador	RF2
Usuario	Usuario
Tipo	Avanzado
Descripción	Los usuarios podrán registrarse e iniciar sesión en la aplicación mediante delegación a terceros de autenticación y autorización con OAuth 2.0 de Google.

Identificador	RF3
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán editar los datos de su perfil así como darse de baja en la aplicación.

Identificador	RF4
Usuario	Usuario
Tipo	Avanzado
Descripción	Los usuarios podrán habilitar o deshabilitar la visibilidad de su perfil. El perfil podrá ser público o privado. La información del perfil público (nombre de usuario, horario y localidad) será visible por cualquier usuario, mientras que en los perfiles privados solo será visible el nombre de usuario.

Identificador	RF5
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán agregar a otros usuarios a su lista de conocidos, permitiendo así la visibilidad de la información del perfil y la asignación a viajes recurrentes.

Identificador	RF6
Usuario	Usuario (Rol Pasajero)
Tipo	Básico
Descripción	Los usuarios podrán buscar un viaje, filtrando por origen, destino, fecha y hora. Además, podrán solicitar plaza en el que se adecúe a sus necesidades.

Identificador	RF7
Usuario	Usuario (Rol Conductor)
Tipo	Básico
Descripción	Los usuarios podrán publicar un viaje especificando origen, destino, fecha, hora y número de plazas disponibles. Este viaje será visible en las búsquedas de los pasajeros mientras tenga plazas disponibles y no haya pasado la fecha y hora de salida.

Identificador	RF8
Usuario	Usuario (Rol Conductor)
Tipo	Básico
Descripción	Los usuarios que ofrezcan plazas en un viaje podrán recibir las peticiones de pasajeros, que podrán ser aceptadas o rechazadas.

Identificador	RF9
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán importar su calendario del curso académico para poder visualizar la hora que comienza la primera y termina la última clase de cada día. Esto se realizará por medio del archivo .ical que se puede generar en el calendario de UACloud.

Identificador	RF10
Usuario	Usuario (Rol Pasajero)
Tipo	Básico
Descripción	El usuario podrá visualizar en su calendario los días que tengan un viaje acordado y buscar un viaje para aquellos que no tengan.

Identificador	RF11
Usuario	Usuario (Rol Conductor)
Tipo	Básico
Descripción	Los usuarios podrán visualizar en el calendario los días en los que ofrecen un viaje y publicar viajes en los días que no tengan.

Identificador	RF12
Usuario	Usuario (Rol Conductor)
Tipo	Básico
Descripción	Los usuarios podrán publicar viajes recurrentes, que se repetirán durante el cuatrimestre, de modo que se publicarán los viajes de ese día de la semana automáticamente.

Identificador	RF13
Usuario	Usuario (Rol Pasajero)
Tipo	Básico
Descripción	Los usuarios podrán solicitar a otro usuario una plaza en los viajes recurrentes.

Identificador	RF14
Usuario	Usuario (Rol Conductor)
Tipo	Avanzado
Descripción	Los usuarios podrán asignar plaza a otros usuarios en los viajes recurrentes.

Identificador	RF15
Usuario	Usuario (Rol Conductor)
Tipo	Básico
Descripción	Los usuarios podrán cancelar el viaje recurrente de una fecha concreta.

Identificador	RF16
Usuario	Usuario (Rol Pasajero)
Tipo	Básico
Descripción	Los usuarios podrán podrán liberar su plaza asignada en los viajes recurrentes en una fecha concreta.

Identificador	RF17
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios dispondrán de un chat por cada viaje, en el que estarán el conductor y demás pasajeros para acordar hora de salida y lugar de encuentro.

Identificador	RF18
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán hacer una valoración por cada viaje sobre el conductor y los demás pasajeros.

Identificador	RF19
Usuario	Usuario
Tipo	Básico
Descripción	Los usuarios podrán activar y desactivar los distintos tipos de notificaciones que realizará la aplicación.

Identificador	RF20
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación notificará diariamente a los usuarios los viajes programados para el día siguiente.

Identificador	RF21
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación notificará a los pasajeros la cancelación de un viaje por parte del conductor.

Identificador	RF22
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación notificará al conductor la petición de una plaza en un viaje ofertado.

Identificador	RF23
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación notificará al pasajero la respuesta a la petición de una plaza en un viaje ofertado.

Identificador	RF24
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación notificará al conductor la liberación de una plaza por parte de un pasajero.

Identificador	RF25
Usuario	Aplicación
Tipo	Básico
Descripción	La aplicación realizará un estudio de los horarios insertados durante las primeras semanas del cuatrimestre para encontrar horarios similares e informará a los usuarios las coincidencias a modo de sugerencia.

Identificador	RF26
Usuario	Administrador
Tipo	Básico
Descripción	El usuario administrador podrá visualizar las estadísticas e indicadores de la aplicación en el panel de control a fin de comprobar el correcto funcionamiento.

Identificador	RF27
Usuario	Administrador
Tipo	Básico
Descripción	El usuario administrador podrá suspender temporal o permanentemente la cuenta de un usuario por hacer un mal uso de la aplicación.

7.3.2. Requisitos no funcionales

Los requisitos no funcionales especifican aspectos importantes que deberá tener la aplicación para conseguir los objetivos marcados. El campo tipo indica el área con el que se relacionan y en todos ellos como usuario al que se dedica el requisito tienen la propia aplicación. El prefijo del identificador es RNF.

Tabla 7.3: Requisitos no funcionales de la aplicación

Identificador	RNF1
Usuario	Aplicación
Tipo	Seguridad
Descripción	El sistema debe implementar seguridad impidiendo el acceso a los datos desde el exterior. Además, debe garantizar una comunicación segura entre cliente y servidor HTTPS con instalación de certificados SSL/TLS.

Identificador	RNF2
Usuario	Aplicación
Tipo	Rendimiento
Descripción	La aplicación debe gestionar de manera correcta las peticiones para garantizar la atención de todas ellas en momentos de mucho tráfico. Los procesos de cálculo de sugerencias o emisión masiva de notificaciones se deben realizar en las horas con menor actividad de peticiones.

Identificador	RNF3
Usuario	Aplicación
Tipo	Usabilidad
Descripción	La aplicación será de uso diario por lo que este debe resultar rápido y cómodo para el usuario.

Identificador	RNF4
Usuario	Aplicación
Tipo	Uso de notificaciones
Descripción	El uso de notificaciones no debe saturar al usuario. Se debe conseguir que estas aporten información al usuario y que le resulten útiles.

Identificador	RNF5
Usuario	Aplicación
Tipo	Disponibilidad
Descripción	La aplicación debe poder ser utilizada desde cualquier navegador. Funcionalidades como la instalación en la pantalla de inicio o el envío de notificaciones deben estar disponibles en todos aquellos dispositivos y navegadores que lo permitan.

8. Diseño

8.1. Diseño de la persistencia

El diseño de la persistencia es fundamental para el desarrollo del proyecto. En este apartado se determinará la información que será almacenada y la manera en que será organizada para poder hacer uso de ella de la manera más eficiente posible. La base de datos será uno de los principales pilares de la aplicación, por lo que cabe dedicar un tiempo considerable a esta etapa del desarrollo ya que un diseño erróneo o incompleto supondría realizar cambios durante etapas posteriores. De esta manera, tras realizar un completo diseño, durante la implementación no saldrán imprevistos o, en caso de salir, serán menos trascendentes.

8.1.1. Diseño de la base de datos

Tal y como se especificó en el apartado de Estado del arte, la base de datos a implementar será relacional. Esto se debe a que la información almacenada tendrá una estructura fija, con unas entidades que tendrán unos atributos fijos. Además, estas entidades mantendrán relación entre ellas, por lo que es conveniente que la base de datos sea estricta en este aspecto para garantizar en todo momento que los datos almacenados se relacionen correctamente.

El conjunto de características de las bases de datos relacionales, resumidas en el acrónimo ACID (Atomicity, Consistency, Isolation and Durability) es una de las prinicipales razones que justifican el uso de este tipo de base de datos. Estas propiedades, en español: Atomicidad, Consistencia, Aislamiento y Durabilidad, garantizan que las transacciones se procesen de manera fiable (*ACID - PiperLab*, 2020).

De manera resumida, la atomicidad supone que los cambios se deben realizar de manera completa o en su defecto no realizarse. La consistencia, por su parte, garantiza que los cambios se realicen desde un estado válido a otro también válido y que cumpla con las restricciones y esquema de datos. El aislamiento de las transacciones permite realizar cambios simultáneos en la base de datos. Y por último, la durabilidad asegura que un cambio completado se mantendrá aunque falle el sistema posteriormente.

Así pues, estas propiedades aportarán al sistema robustez y menor vulnerabilidad ante fallos (*Bases de datos relacionales vs. no relacionales: ¿qué es mejor? - Aukera*, 2018), garantizando que los datos almacenados sean válidos y estructurados de la manera deseada.

Para realizar el diseño de la base de datos, se seguirá la metodología empleada en la asignatura Diseño de Base de Datos Multimedia del Grado en Ingeniería Multimedia de la Universidad de Alicante. Esta manera de diseñar la base de datos permite convertir una descripción de los requerimientos estructurales y relacionales del sistema de almacenamiento a un diseño relacional exacto, necesario para implementar la base de datos.

A continuación se detallarán los resultados obtenidos en cada una de las fases de esta metodología.

8.1.2. Descripción del sistema de almacenamiento.

La descripción del sistema de almacenamiento de datos es la primera fase. En ella se explicarán los datos que se almacenarán y las relaciones entre ellos. El sistema tendrá 2 entidades principales: usuarios y viajes.

Por lo que respecta a los usuarios, de estos se almacenará un identificador, email, nombre de usuario, contraseña de acceso (cifrada por seguridad), nombre, apellidos, edad, foto de perfil y la suscripción que permitirá enviar notificaciones a su dispositivo en caso de permitirlas. Inicialmente el usuario no estará verificado hasta que confirme su identidad accediendo a un enlace enviado a su correo electrónico. Además, el usuario podrá indicar la localidad donde reside y la universidad donde estudia. Estos dos últimos campos serán opcionales, como también lo serán los relativos al nombre, apellidos, edad y foto de perfil.

Los usuarios podrán establecer relación entre ellos de modo que un usuario puede tener una lista de conocidos. Para establecer esta relación de conocidos, esta debe ser aprobada por ambos, uno realizando la petición y el otro aceptándola.

Para tener control sobre el calendario y horarios de los usuarios, se almacenarán los días que estos tienen clase. Como información complementaria de cada día de clase se guardará la hora de inicio y la hora de finalización.

Respecto a las notificaciones, se deberá tener un registro del tipo de notificaciones que desea recibir cada usuario. Cada tipo tendrá un mensaje base sobre el cual se añadirán los datos específicos completando así el mensaje de la notificación a enviar. Además, el sistema debe poder almacenar una lista de notificaciones para poder ser enviadas de forma masiva a los usuarios.

Por otra parte, respecto a los viajes, de estos se almacenará un identificador, sentido del viaje (ida o vuelta), número de plazas ofrecidas, la fecha y hora de salida y llegada. Según el sentido, será necesario almacenar una u otra fecha y hora, siendo posible especificar ambas en cualquier sentido. Los viajes deberán tener necesariamente un origen y un destino.

Este origen y destino de los viajes dependerá también del sentido del viaje. Pues en el viaje de ida el origen será una localidad y el destino una universidad, mientras que el los viajes de

vuelta será al revés. Las universidades, además, estarán ubicadas en una localidad.

Cada viaje será ofrecido necesariamente por un usuario y los demás usuarios podrán pedir plaza en él. El usuario que ofrece el viaje podrá aceptar o declinar la solicitud, por lo que deben estar registradas las solicitudes y su estado.

Los usuarios también podrán registrar viajes recurrentes. De estos se almacenará el día de la semana, el sentido, el origen y el destino. Estos viajes recurrentes, generarán una lista de viajes. Por esta razón, un viaje puede pertenecer a un viaje recurrente. Los usuarios podrán suscribirse a un viaje recurrente. Esta suscripción deberá ser aceptada o declinada por el usuario que ofrece el viaje recurrente.

Con el fin de tener un registro de las conversaciones de cada viaje, en el sistema se almacenarán los mensajes. Un mensaje pertenecerá a un viaje concreto y estará escrito por un usuario. Del mensaje se guardará, además, el contenido del mismo y la fecha y hora en que se envía.

Por último, respecto a las valoraciones que pueden hacer los usuarios sobre los usuarios que comparten viaje, de cada valoración se almacenará el usuario que la realiza, la puntuación, el usuario al que se valora, el rol sobre el que se valora (pasajero o conductor) y un posible comentario.

8.1.3. Esquema Entidad-Relación

La segunda fase del diseño consiste en representar las exigencias y requerimientos descritos anteriormente en un esquema Entidad-Relación. Este permite visualizar de manera clara las diferentes entidades que formarán parte de la base de datos y las relaciones que existen entre ellas. En la figura 8.1 se puede visualizar el esquema obtenido.

8.1.4. Modelo relacional de la Base de Datos

Como última fase, partiendo del esquema Entidad-Relación, se obtendrá el modelo relacional de la base de datos. En este se representarán cada una de las tablas que tendrá la base de datos y los atributos de cada una de ellas. Las claves de estas tablas vendrán determinadas en función del tipo de relación entre las entidades. A continuación, se muestra el formato que será seguido para especificar cada tabla:

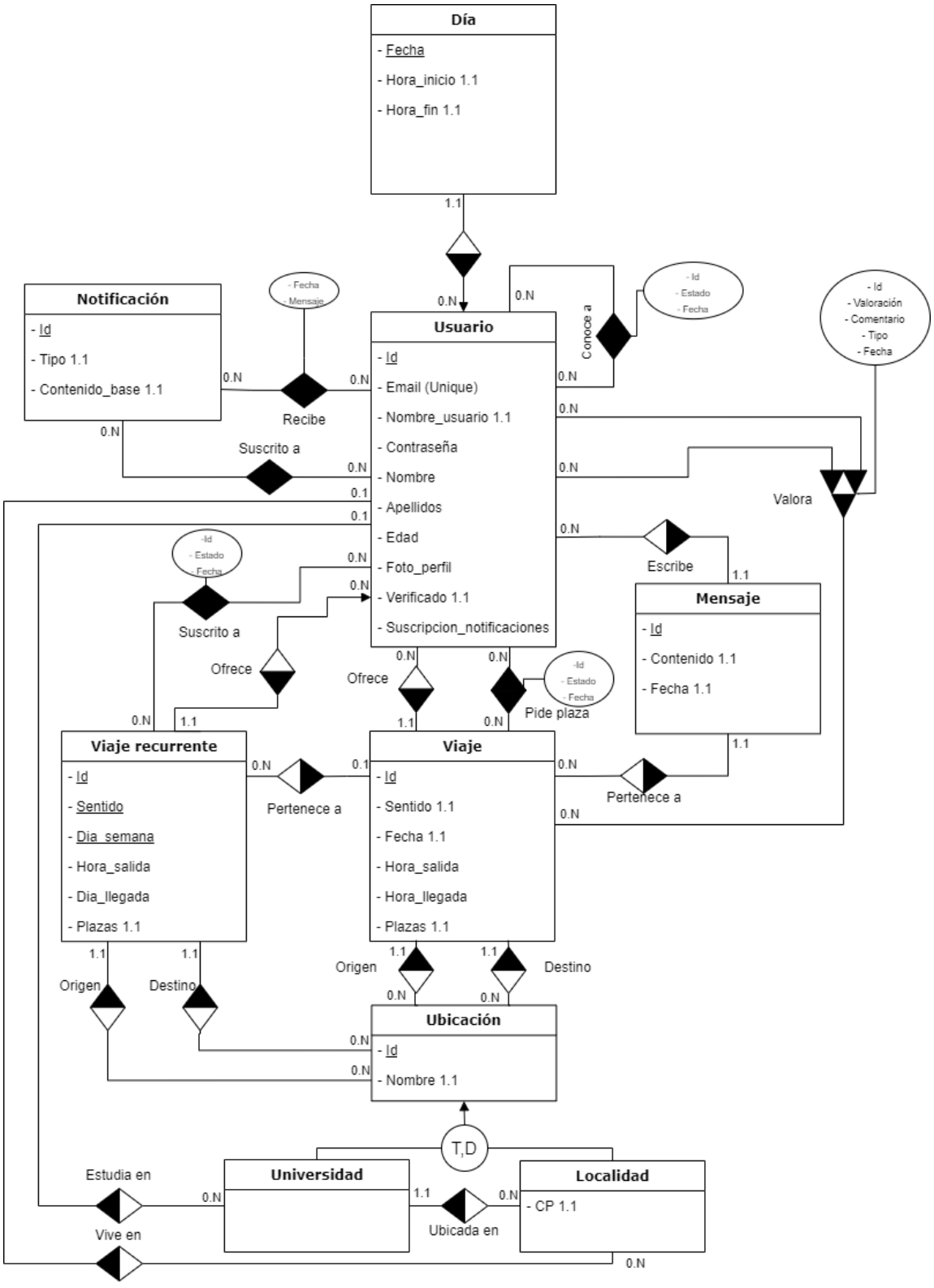


Figura 8.1: Esquema Entidad-Relación de la Base de Datos

NOMBRE_TABLA atributo1 (tipo), atributo2 (tipo), ... Lista de atributos de la tabla a almacenar
CP (atributo1, atributo2, ...) Atributos que forman la clave primaria, no podrán ser duplicados ni nulos
C. Alt. (atributo1, atributo2, ...) Atributos que forman una clave alternativa, no podrán ser duplicados ni nulos
C.Aj. (atributos) -> TABLA_DESTINO Atributos que harán referencia a la clave primaria de otra tabla
V.N.N atributo1, atributo2, ... Atributos que no nulos

El conjunto de tablas obtenido del esquema Entidad-Relación es el siguiente:

Tabla 8.1: Modelo relacional de la Base de Datos

USUARIO id (int), email (string), nombre_usuario (string), contraseña (string), nombre (string), apellidos (string), edad (int), foto_perfil (string), verificado (boolean), suscripcion_notificaciones (string), vive_en (int), estudia_en (int)
CP (id)
C.Alt. (email)
C. Aj. (vive_en) -> LOCALIDAD
C. Aj. (estudia_en) -> UNIVERSIDAD
V.N.N id, email, nombre_usuario, contraseña

DIA usuario (int), fecha (date), hora_inicio (datetime), hora_fin (datetime)
CP (usuario, fecha)
C.Aj. (usuario) -> USUARIO
V.N.N hora_inicio, hora_fin

CONOCE_A usuario1 (int), usuario2 (int), fecha (datetime), estado (int)
CP (usuario1, usuario2)
C.Aj. (usuario1) -> USUARIO
C.Aj. (usuario2) -> USUARIO
V.N.N fecha estado

NOTIFICACION id (int), tipo (string), contenido_base (string)
CP (id)
V.N.N tipo, contenido_base

NOTIFICACIONES_PROGRAMADAS usuario (int), tipo_notificacion (int), mensaje (string), fecha (datetime)
CP (usuario, tipo_notificacion, fecha)
C.Aj. (usuario) -> USUARIO
C.Aj. (tipo_notificacion) -> NOTIFICACION
V.N.N mensaje

SUSCRITO_A usuario (int), tipo_notificacion (int)
CP (usuario, tipo_notificacion)
C.Aj. (usuario) -> USUARIO
C.Aj. (tipo_notificacion) -> NOTIFICACION

VIAJE id (int), conductor (int), origen (int), destino (int), sentido (int), fecha (date), hora_salida (time), hora_llegada (time), recurrente (int)
CP (id)
C.Aj. (conductor) -> USUARIO
C.Aj. (origen) -> UBICACIÓN
C.Aj. (destino) -> UBICACIÓN
C.Aj. (recurrente) -> VIAJE_RECURRENTE
V.N.N conductor, origen, destino, fecha, sentido

VIAJE_RECURRENTE id (int), conductor (int), origen (int), destino (int), sentido (int), dia_semana (int), hora_salida (time), hora_llegada (time)
CP (id)
C.Alt. (conductor, dia_semana, sentido)
C.Aj. (conductor) -> USUARIO
C.Aj. (origen) -> UBICACIÓN
C.Aj. (destino) -> UBICACIÓN
V.N.N conductor, origen, destino, sentido, hora_salida, hora_llegada

UBICACION id (int), nombre (string)
CP (id)
V.N.N nombre

LOCALIDAD ubicacion (int), codigo_postal (string)
CP (ubicacion)
C.Aj. (ubicacion) -> UBICACIÓN
V.N.N codigo_postal

UNIVERSIDAD ubicacion (int), localidad (int)
CP (ubicacion)
C.Aj. (ubicacion) -> UBICACIÓN
C.Aj. (localidad) -> LOCALIDAD
V.N.N localidad

PLAZA usuario (int), viaje (int), fecha (datetime), estado (int), id (int)
CP (usuario, viaje)
C.Alt (id)
C.Aj. (usuario) -> USUARIO
C.Aj. (viaje) -> VIAJE
V.N.N usuario, viaje, fecha, estado

PLAZA_REC usuario (int), viaje_recurrente (int), fecha (datetime), estado (int), id (int)
CP (usuario, viaje_recurrente)
C.Alt (id)
C.Aj. (usuario) -> USUARIO
C.Aj. (viaje) -> VIAJE_RECURRENTE
V.N.N usuario, viaje, fecha, estado

MENSAJE id (int), usuario (int), viaje (int), contenido (string), fecha (datetime)
CP (id)
C.Aj. (usuario) -> USUARIO
C.Aj. (viaje) -> VIAJE
V.N.N usuario, viaje, contenido, fecha

VALORACION usuario (int), valorado (int), viaje (int), valoracion (int), comentario (string), fecha (datetime), tipo (int), id (int)
CP (usuario, valorado, viaje)
C.Alt (id)
C.Aj. (usuario) -> USUARIO
C.Aj. (valorado) -> USUARIO
C.Aj. (viaje) -> VIAJE
V.N.N valoracion, fecha, tipo

8.2. Diseño arquitectura conceptual

La arquitectura conceptual es la que define, a grandes rasgos, la estructura del proyecto. Más allá de las tecnologías utilizadas, que se expondrán más adelante, en este apartado se explicarán las partes que componen la aplicación.

Así pues, el proyecto será desarrollado atendiendo a la definición del estilo de programación por capas, donde el objetivo primordial es la separación (desacoplamiento) de las partes que componen una arquitectura cliente-servidor: capa de presentación, lógica del negocio, y capa de datos. (*Programación por capas*, 2014)

Este modelo de desarrollo ofrece múltiples ventajas. Entre ellas, el mencionado desacoplamiento entre partes o capas, lo que permite realizar cambios en cualquier parte sin que esto afecte directamente a las demás. Además, este modelo facilita la distribución de las capas en diferentes entornos de ejecución, que se conectan mediante API's. Por esta razón es posible incluir diferentes alternativas en la capa de presentación, como sería una posible aplicación nativa futura, la cual podría compartir capa de datos con la aplicación web.

Cabe destacar, en este contexto, el término "nivel" para entender el modelo. Este corresponde a la forma en que las capas están distribuidas físicamente. Por ejemplo, en una solución a tres niveles, cada capa residiría en un ordenador o servidor, mientras que en una solución a dos niveles, la capa de presentación y datos estarían en diferentes ordenadores, pudiendo estar la capa de lógica en una de las dos o repartida entre ambas.

En la figura 8.2 se puede visualizar el esquema conceptual con las diferentes partes o niveles. Se trata de una solución de tres capas en dos niveles. La distribución de las capas es: presentación+lógica por un lado y lógica+datos por el otro lado. La conexión entre ambas se realizará mediante una API con autenticación y verificación de usuario basada en token.

Aunque físicamente no se trabajará con dos servidores diferentes, esta separación en niveles

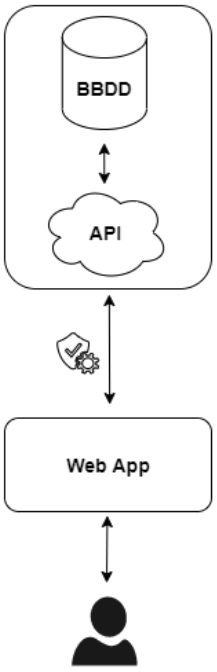


Figura 8.2: Esquema de la arquitectura conceptual

será posible gracias a las tecnologías Docker y CapRover, explicadas en el apartado Estado del arte, que permiten trabajar con varios entornos de ejecución simultáneamente, distinguiendo así los dos niveles.

El primer nivel, la propia aplicación web, será la que permitirá la interacción del usuario mediante la capa de presentación. La capa lógica en este nivel se encargará de gestionar las peticiones realizadas por el usuario para consultar o insertar los datos en la capa de datos del segundo nivel. En el segundo nivel, en cambio, se atenderán las peticiones de datos realizadas en la aplicación web. Además esta capa tendrá cierta lógica ya que la propia API, como se mencionará más adelante en la sección Diseño API Rest, tendrá rutas específicas más allá de las CRUD. El acrónimo CRUD, en inglés, Create, Read, Update and Delete, hace referencia a los funciones básicas en la capa de datos de una aplicación, que en español significan: Crear, Leer, Actualizar y Borrar.

8.3. Diseño API Rest

En el proyecto, la API Rest será la encargada de permitir la conexión entre las diferentes capas. Un buen diseño de esta ayudará en gran medida a conseguir un bajo acoplamiento entre las partes y esto, a su vez, permitirá conectar capas de presentación alternativas e independientes.

Para obtener las rutas básicas de la API, se tendrán en cuenta las entidades del Esquema Entidad-Relación de la Base de Datos en la sección Diseño de la persistencia. Cada una de las entidades tendrá su propio CRUD, a excepción de las ubicaciones, las cuales serán fijas.

Sin embargo, la API, no se limitará a realizar estas operaciones básicas CRUD. Además, tendrá rutas específicas que reducirán el trabajo de la capa lógica de negocio. Para obtener un listado completo de las peticiones que deberá atender la API, se tendrá en cuenta el listado de interfaces de la sección Diseño Interfaces, de modo que analizando las interacciones en la interfaz surgirán todas las peticiones a contemplar.

El listado completo de peticiones está incluído en el anexo Diseño de API

8.4. Diseño de la arquitectura tecnológica Front/Back-end

Tras una primera definición de los diferentes niveles y capas del proyecto en la sección de Diseño arquitectura conceptual, a continuación se expondrán las tecnologías que se emplearán para implementarlos.

En la figura 8.3 se pueden visualizar las tecnologías empleadas para cada una de las capas y niveles.

En primer lugar, con el objeto de proporcionar una alternativa al registro mediante email en la aplicación, se utilizará la autentificación con OAuth 2.0 concretamente de Google. El acceso a la aplicación se podrá realizar bien mediante navegador, tanto en smartphone como en ordenador, o bien mediante la PWA instalable en los dispositivos.

A continuación, en el primer nivel, la aplicación web, con respecto a la capa de presentación se emplearán los lenguajes de programación HTML, CSS y Javascript. Estos además serán complementados con frameworks y bibliotecas de código abierto, como son AngularJS, JQuery y Bootstrap, que agilizarán el proceso de desarrollo.

El framework AngularJS se utilizará para diseñar la aplicación web en una sola página, basada en el patrón MVC (Modelo - Vista - Controlador). Para simplificar la interacción con los elementos del DOM, manejo de eventos y animaciones se empleará la biblioteca JQuery. Y finalmente, para facilitar la implementación del estilo se hará uso de la biblioteca Bootstrap.

En este mismo nivel, pero gestionando la capa lógica del negocio, será empleado NodeJS. Este contará con diferentes librerías específicas, también de código abierto, para cubrir todas las funcionalidades de la aplicación. Estas librerías son proporcionadas por el gestor de paquetes NPM de NodeJS.

La gestión de las Notificaciones Push se realizará con la librería web-push del gestor de paquetes NPM. Otra de las librerías empedadas será socket.io, la cual permitirá la comunica-

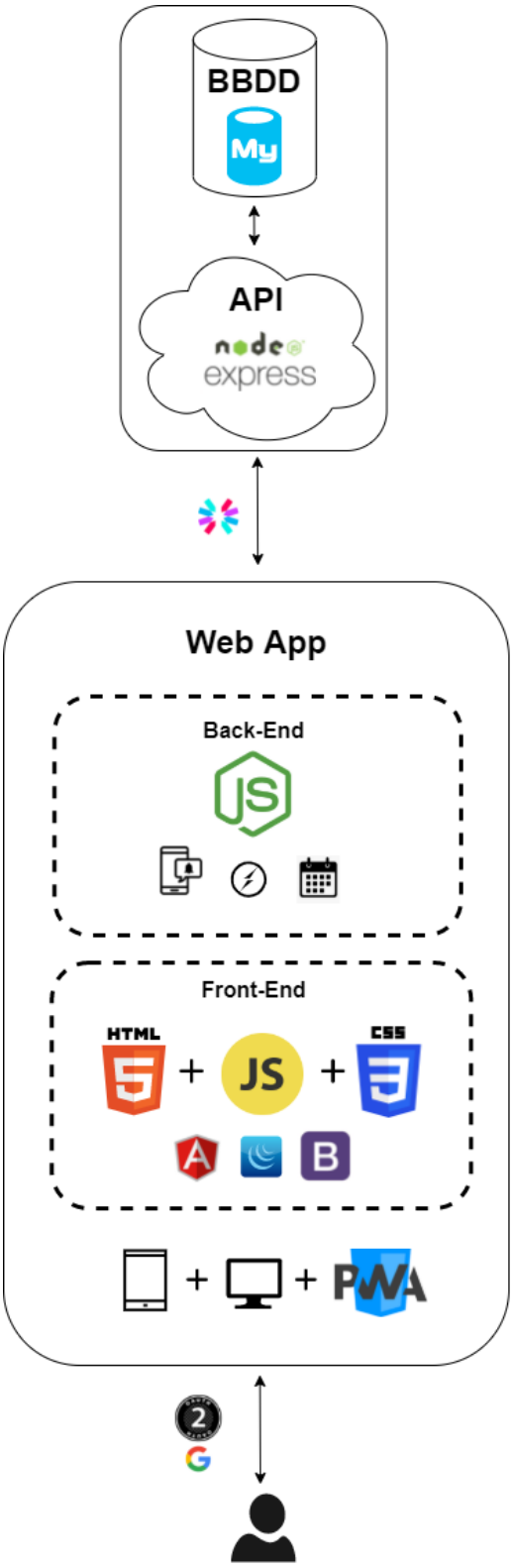


Figura 8.3: Esquema de la arquitectura tecnológica

ción bidireccional en tiempo real entre el cliente y el servidor web, por ejemplo en los chats. Por último, la libreria ical.js permitirá la importación de los calendarios.

La conexión entre ambos niveles será controlada mediante autorización basada en token. Así pues, será JWT (JSON Web Token) el estándar empleado para permitir la propagación de identidad y privilegios.

Ya en el segundo nivel, la capa lógica también estará implementada con NodeJS. Con la librería Express se especificarán las rutas de la API. Y finalmente en la capa de datos, para su almacenamiento se hará uso de una base de datos relacional gestionada con MySQL.

8.5. Diseño de la Interacción o Experiencia de Usuario

Analizar cuál será el comportamiento del usuario durante el uso de la aplicación es también un paso trascendental antes de la etapa de implementación. El primer factor a tener en cuenta es que el usuario no utilizará la aplicación por gusto. La necesidad de cubrir una carencia es lo que llevará al usuario a buscar una aplicación que cumpla con una determinada función.

Por esta razón, es necesario empatizar con el usuario para realizar un diseño adaptado a él. Esto conllevará una buena primera impresión por su parte y, por tanto, una mayor probabilidad de que elija nuestra aplicación por delante de las alternativas.

A continuación, serán dos los tipos de análisis realizados con este objetivo: el diseño de personas y la reacción de estas al uso de la aplicación.

8.5.1. Diseño de Personas

El diseño de Personas consiste en definir usuarios potenciales de la aplicación. A pesar de ser personajes ficticios, sus estilos de vida podrían compararse con los de una persona real.

Estas descripciones permiten ajustar el diseño de la aplicación, así como la inserción de nuevas funcionalidades, con el objetivo de cubrir las necesidades de estos prototipos de usuario.

Para realizar estas descripciones se ha seguido un artículo sobre la experiencia de usuario publicado en la web de Qubstudio, un equipo de estrategas y analistas de negocio. (*4 Examples of UX Personas*, 2021)

LUIS

Información básica

Tiene 21 años y vive con sus padres en Castalla, a 25 minutos en coche de la universidad. Tiene carné de conducir y coche propio.

Ocupación y aficiones

Está en 3º de Ingeniería Multimedia en la Universidad de Alicante. Su horario semanal es fijo, por lo que se siente cómodo en su rutina. A Luís le gusta conducir y no le importa viajar con su coche a diario para ir a la universidad.

Objetivos y necesidades

No tiene una fuente de ingresos propia, por lo que sus gastos van a cargo de sus padres. Por ello, Luís considera necesario reducir el gasto en combustible.

Preocupaciones

A Luís no le gusta depender de los horarios establecidos por los medios de transporte públicos. Además, considera que pierde demasiado tiempo tanto en las esperas como durante los viajes en este medio.

Patrones de comportamiento

Amante de las nuevas tecnologías, Luís considera que el uso de estas es fundamental actualmente. No duda en incluir su smartphone u ordenador como complemento en sus hábitos diarios.

ANA

Información básica

Tiene 20 años y vive con sus padres en Almoradí, a 35 minutos en coche de la universidad. No tiene carné de conducir.

Ocupación y aficiones

Está en 2º de Enfermería en la Universidad de Alicante. Es una chica muy organizada, necesita definir sus planes con días o semanas de antelación. Sin embargo, su horario semanal variante en la universidad complica su planificación.

Objetivos y necesidades

A pesar de haber estudiado la posibilidad de alquilar un piso o residencia cerca de la universidad, debido a sus condiciones económicas familiares, Ana no puede permitírselo. Está obligada, por tanto, a desplazarse diariamente.

Preocupaciones

Los horarios de los autobuses que viajan desde su localidad a la universidad no se adaptan a los suyos. Además, no quiere correr el riesgo de perder el autobús y llegar tarde a clase.

Patrones de comportamiento

Ana suele utilizar una agenda para tener organizados todos sus planes. En ella, entre otras muchas cosas, apunta todo lo relativo a los horarios de dichos planes, que considera muy importante cumplir.

JUAN

Información básica

Tiene 18 años y vive con sus padres en Villena, a 35 minutos en coche de la universidad. Tiene carné de conducir pero no dispone de coche propio, lo comparte con su hermano.

Ocupación y aficiones

Está en 1º de CAFD en la Universidad de Alicante. Su horario en la universidad varía según la semana. Le encanta el deporte, que practica a diario y encuentra, en cada momento libre, la oportunidad de realizar cualquier actividad física.

Objetivos y necesidades

Juan no se considera todavía preparado para vivir de forma independiente, por lo que descarta alquilar un piso o residencia universitaria. Necesita encontrar la forma de ir a la universidad cada día.

Preocupaciones

Su falta de experiencia con el coche todavía no le inspira confianza para usarlo habitualmente para ir a la universidad. A pesar de tener la posibilidad de desplazarse en autobús, Juan considera que paga demasiado por un servicio que cumple mínimamente con su función. Le gustaría no perder tanto tiempo cada día, ya que cree que es tiempo que podría dedicar a sus aficiones.

Patrones de comportamiento

Juan no es una persona demasiado previsora. Se podría decir que su estilo de vida se basa en la improvisación, aprovechando cada instante. Por esta razón, en muchas ocasiones, su despreocupación le ha jugado malas pasadas.

Una vez terminada la descripción de las Personas, podemos sacar conclusiones que coincidan en todos ellos.

En primer lugar, podemos apreciar que los tres necesitan desplazarse diariamente a la universidad. Además, por diferentes razones, ninguno está satisfecho con la opción de desplazarse con transporte público. Esto directamente los convierte en usuarios potenciales.

A esta necesidad de desplazarse, preferiblemente en coche, podemos añadir que a los tres les interesaría compartir coche. Cada uno por una razón diferente pero, al fin y al cabo, los tres verían atractiva la idea de viajar con otros estudiantes y por tanto, tendrían en la aplicación la forma de encontrar compañeros de viaje.

Incluso hay un tercer aspecto que podría hacer aún más atractiva la aplicación para estos usuarios. Se trata de la planificación. Ya sea para tener organizados los desplazamientos a la universidad o por la posibilidad de incluir recordatorios sobre los viajes planificados.

8.5.2. User Journey Maps

Tras obtener los prototipos de usuarios de la aplicación, es momento de prever cuál será su experiencia de uso. Para ello se empleará la herramienta conocida como User Journey Map. Traducido como "Mapa del usuario", esta herramienta permite analizar las emociones, sentimientos o impresiones del usuario en cada etapa durante el uso de la aplicación. (*Customer Journey Map o Mapa de Experiencia del Cliente + Ejemplo y Vídeo*, 2021)

Como resultado se obtiene una gráfica que valora el grado de satisfacción del usuario con la acción que está realizando en la aplicación. En el eje X encontramos dichas acciones, mientras que en el eje Y, las partes positiva y negativa corresponden a las respectivas experiencias.

La tendencia de la gráfica será la que nos alertará de los puntos en los que habrá que tener especial atención, pues aquellas acciones que por ser más tediosas o por poder causar una reacción negativa del usuario requerirán más tiempo para identificar el inconveniente y aplicar una solución que minimice el impacto.

En la aplicación son muchas las interacciones que se pueden analizar pero, a continuación, serán dos las analizadas, siendo estas las dos principales. Se trata de buscar y publicar un viaje, en las figuras 8.4 y 8.5 respectivamente.

Por lo que respecta a los pasos para buscar un viaje, encontramos que los dos primeros no son positivos, pues rellenar un formulario, para aplicar los filtros, puede resultar tedioso y más aún lo es más comparar entre los viajes que coincidan con los filtros. Una vez encontrado el viaje, todo resulta positivo excepto en el caso de que la solicitud de plaza a este sea negativa, lo que obligaría a repetir el proceso.

Para solventar estos pasos más costosos y agilizar así el proceso de tratar con los filtros, el formulario podría estar ya completo con la localidad y universidad por defecto del usuario y la fecha del mismo día o la del día siguiente, en caso de haber pasado ya la hora de vuelta de la universidad. Para comparar, los resultados deben estar ordenados de forma que los primeros sean los más interesantes, bien sea por tener conductores conocidos o bien por tener los conductores una buena valoración. Para evitar repetir el proceso de búsqueda al rechazar una solicitud, aunque solo se pueda realizar un viaje de ida y uno de vuelta por día, se podría solicitar más de uno. De esta manera, cuando una de las solicitudes se acepte, se cancelarían

automáticamente las demás pendientes para ese día.

En cuanto al proceso de publicar un viaje, el primer paso, la introducción de los datos, sería comparable a los filtros de la búsqueda ya que se trata de rellenar un formulario. El siguiente momento negativo a considerar, aunque no tenga una interacción directa del usuario, es el hecho de esperar solicitudes, que puede resultar frustrante en caso de que no lleguen. Finalmente, rechazar las solicitudes pendientes en caso de haber ya ocupado todas las plazas sería también un paso tedioso.

Al igual que en la búsqueda, el formulario debería estar rellenado por defecto con los datos del siguiente viaje más próximo en el calendario. El proceso de esperar solicitudes, por no tener interacción directa del usuario sería más complicado de agilizar. Una opción sería notificar la publicación del viaje a los usuarios que estén esperando un viaje con esas características. Y también de forma similar al anterior proceso, cuando se completen las plazas de un viaje, las solicitudes aún pendientes se cancelarían automáticamente.

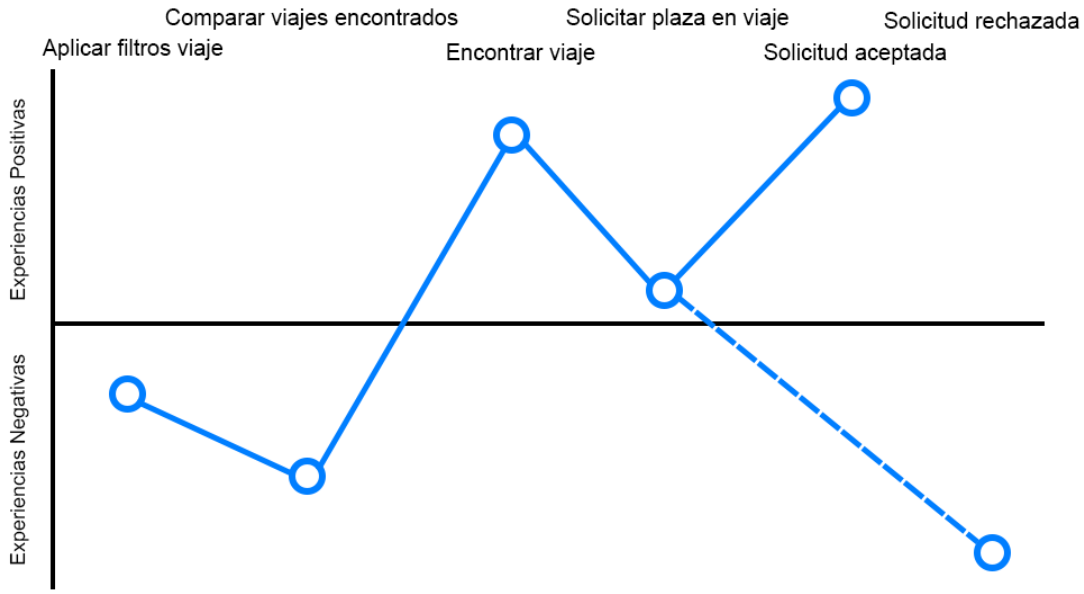


Figura 8.4: Journey map buscar viaje

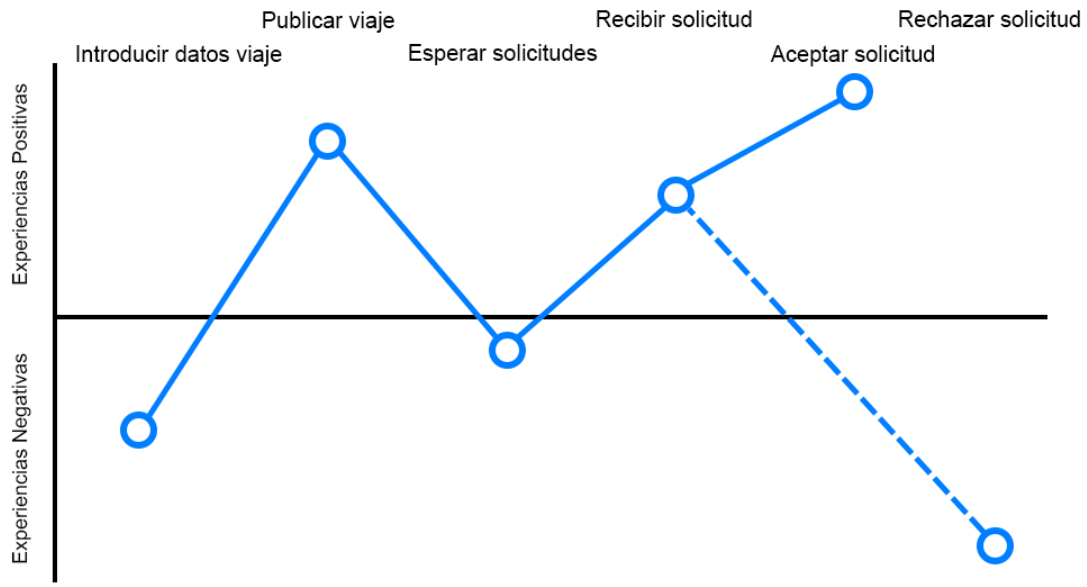


Figura 8.5: Journey map publicar viaje

8.6. Diseño Interfaces

El diseño de interfaces es la parte donde los diferentes requisitos especificados en el apartado de Análisis y especificación se transforman en los mecanismos que permitirán llevar a cabo las funcionalidades.

De esta manera, los requisitos que se caractericen por tener interacción del usuario, estarán estrechamente relacionados con una o más interfaces. En el anexo Diseño de Interfaces, se exponen los bocetos de todas las interfaces, junto a los requisitos que cubren, y una explicación de las partes que la componen.

La representación realizada no es de alta fidelidad, pues la herramienta empleada para ello es Balsamiq. Esta herramienta permite la creación de wireframes y en estos el objetivo es definir las propias funcionalidades contemplando aspectos de usabilidad y accesibilidad. Sin embargo, a partir de esta representación, juntamente con las Guías de estilos expuestas en el siguiente apartado, se podrá definir el aspecto final de la aplicación durante el desarrollo.

La figura 8.6 representa un diagrama de la navegación entre las diferentes interfaces. Se puede observar como, tras el inicio de sesión, la vista principal será la de “Mis viajes”, desde la cual se podrá navegar hasta las demás.

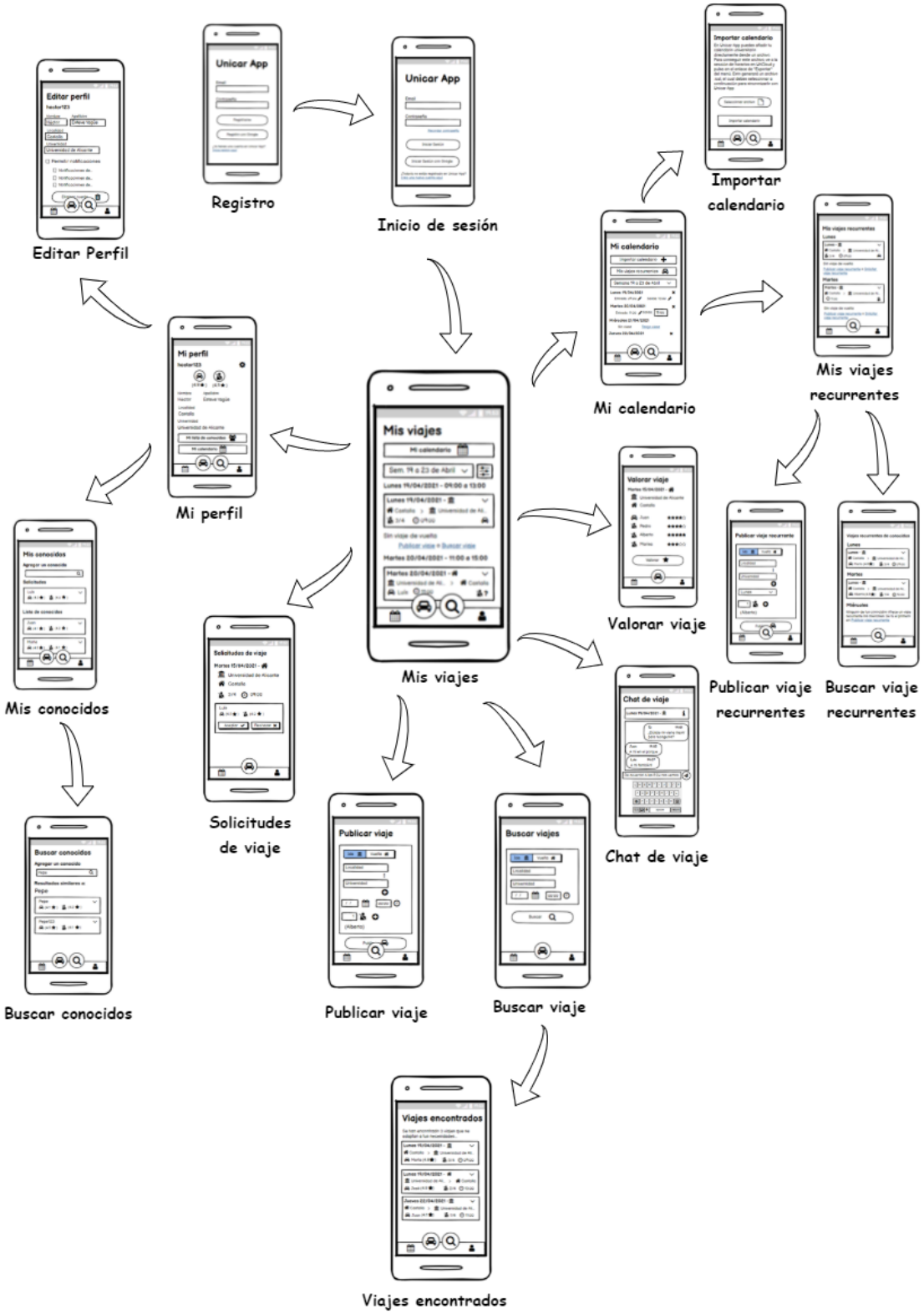


Figura 8.6: Navegación entre interfaces
Fuente propia

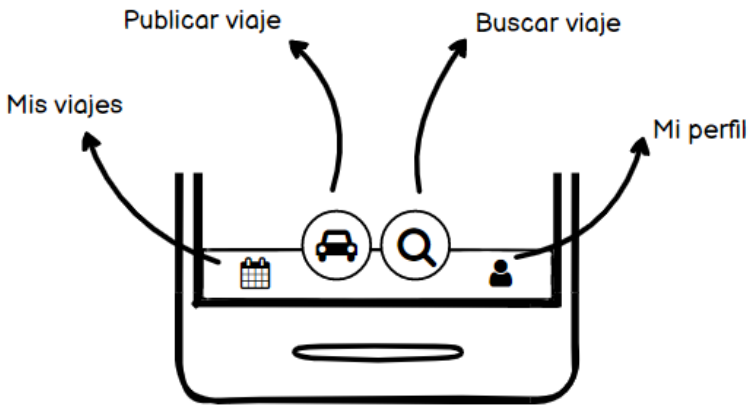


Figura 8.7: Boceto menú

Cabe destacar el menú presente en la parte inferior de todas las interfaces, cuya función es ofrecer al usuario atajos para poder navegar rápidamente a las 4 principales vistas. En la Figura 8.7, de derecha a izquierda, son: Mis viajes, Publicar viaje, Buscar viaje y Mi perfil.

8.7. Guías de estilos

La guía de estilo marcará las pautas a seguir para el diseño de la web. Por tanto, servirá de consulta para visualizar los objetivos de la aplicación en cuanto a su estilo. El principal objetivo es dotar al estilo de la aplicación de la máxima sencillez posible, a fin de que al usuario le resulte intuitivo el uso de la aplicación.

A continuación se señalarán aspectos relativos al estilo como son el logotipo, los colores principales y secundarios, la tipografía y la composición de las interfaces.

8.7.1. Logotipo

El logotipo es el símbolo que representa la temática de la aplicación. Esta imagen podrá ser presentada de diferentes formas, como son el imagotipo y el isotipo. Además, se mostrarán dos versiones de cada representación: una en color y otra en blanco y negro.

Por lo que respecta al isotipo, en la figura 8.8, este se caracteriza por no tener ningún texto. El icono representativo se caracteriza por estar compuesto por dos elementos. El principal es un coche con 5 pasajeros, simbolizando el hecho de compartir coche, mientras que el secundario se trata del icono representativo de la ubicación con un birrete, que simboliza el hecho de encontrar estudiantes.

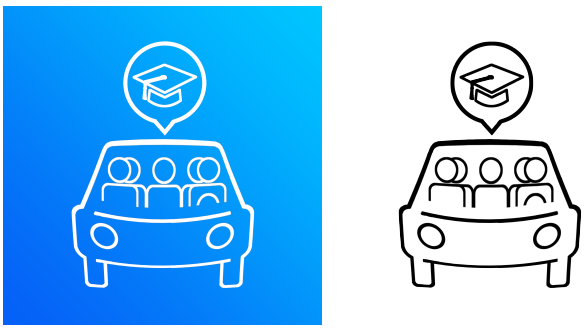


Figura 8.8: Isotipo en color y blanco y negro
Fuente propia

El imagotipo, en cambio, incluye también el texto con el nombre de la marca y está incluido en la figura 8.9.

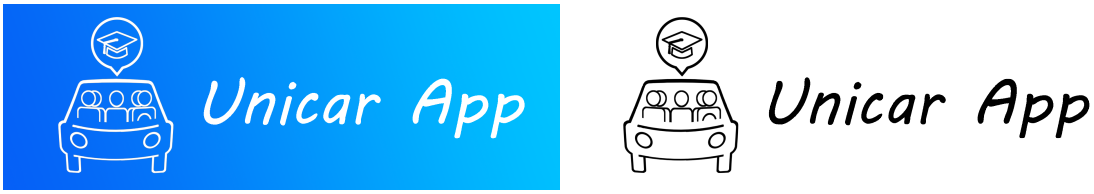


Figura 8.9: Imagotipo en color y blanco y negro
Fuente propia

En ambas representaciones y concretamente en las versiones en color de estas, los colores empleados para degradado del fondo han sido los corporativos, que se detallarán a continuación.

8.7.2. Guía de colores

Los colores corporativos de la aplicación componen una gama cromática fría, orientada a tonos azules.

En primer lugar, los colores corporativos que se utilizarán mayoritariamente en el diseño de las interfaces están reunidos en la paleta de la figura 8.10. Por una parte, “Bluetiful” y “Vivid Sky Blue” serán los colores de los elementos con los que el usuario podrá interactuar, como botones y enlaces. Y, por otra parte, “Black Chocolate” y “Azyre X Web Color” serán los correspondientes al texto y fondo de la aplicación.

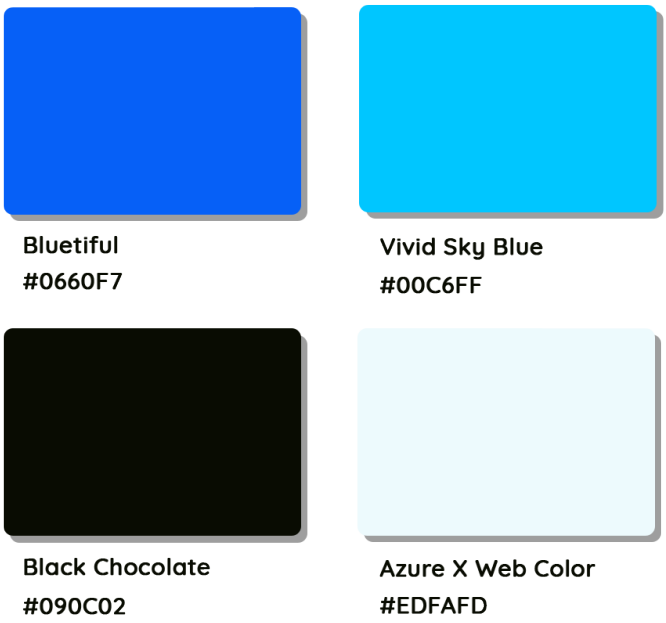


Figura 8.10: Paleta de colores principales
Fuente propia

No obstante, a estos colores principales se les añadirán 3 más que estarán relacionados con la acción que realizan los botones en los que se apliquen. Se trata del color "Rufous" para acciones como "Eliminar" y "Cancelar", el "Maximum Yellow" para "Editar" y "Leaf Green" para "Enviar" y "Aceptar".

Figura 8.11: Paleta de colores secundarios
Fuente propia

Cabe mencionar que la combinación de colores se ha obtenido mediante la web *Coolors - The super fast color schemes generator!* (2021). Esta web permite generar combinaciones de colores partiendo de colores introducidos por el usuario. Para este caso, se han introducido los colores principales, "Bluetiful" y "Vivid Sky Blue", y la propia web ha generado el resto de colores de las paletas.

8.7.3. Tipografía

La tipografía utilizada es un tipo de letra sencilla y universal que facilite la legibilidad. Se ha optado por emplear una fuente gratuita de Google Fonts, como es Roboto. En la figura 8.12 se incluye una representación de dicha fuente con los caracteres más comunes.

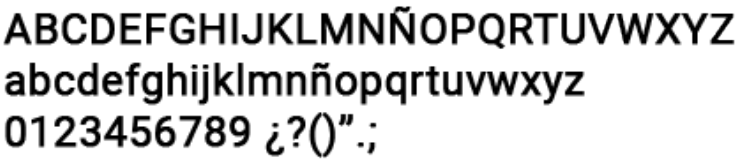


Figura 8.12: Caracteres tipografía Roboto
Fuente propia

8.7.4. Composición de las interfaces

La composición de las interfaces expuesta a continuación corresponde a la versión adaptada para móvil, que será la versión implementada en primer lugar. La adaptación para pantallas de mayor tamaño se realizará en un futuro.

Distinguiremos entre 3 tipos de interfaces según su composición: lista, tarjeta y formulario. La comparación entre ellas puede observarse en la figura 8.13

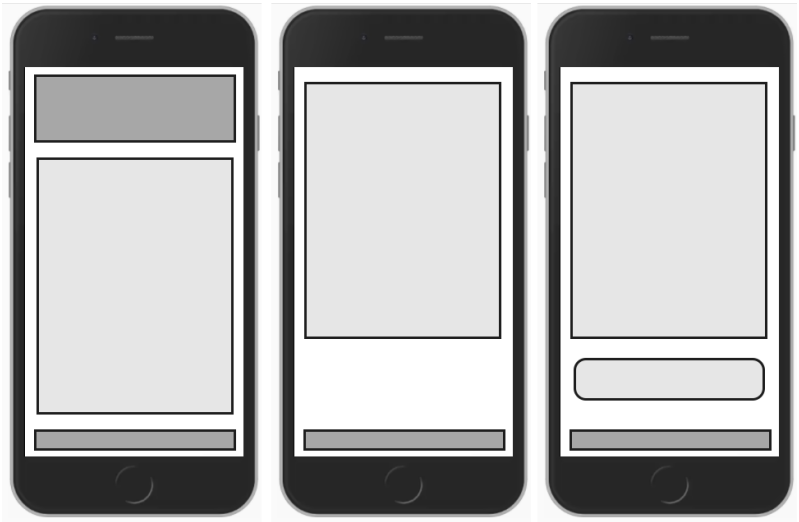


Figura 8.13: Composición interfaces (de izquierda a derecha): lista, tarjeta y formulario
Fuente propia

Todas ellas tienen un único elemento en común: el menú de navegación de la parte inferior. Este elemento tiene una posición fija por lo que siempre estará en dicha ubicación.

En el caso de la composición de las interfaces de tipo lista, estas tienen, además, otro menú superior que incluirá también enlaces para navegar a otras páginas. Este menú también estará fijo siempre en la parte superior. Sin embargo, en la parte central y principal de esta, los elementos de la lista se podrán desplazar verticalmente para poder visualizarlos todos.

Las otras dos composiciones comparten el elemento principal que es la tarjeta con el contenido de la vista. No obstante, la composición de los formularios contendrá debajo de la tarjeta, el botón para enviar el contenido del mismo.

8.8. Diseño de pruebas y validación

Una vez obtenida una versión estable de la aplicación con las principales funcionalidades implementadas por completo, la aplicación se desplegará en el entorno de pruebas, descrito en el apartado de Estado del arte. Esta versión será compartida a un reducido grupo de personas de confianza para que prueben dicha versión.

Estas funcionalidades principales, que se le indicarán a dichos usuarios, son: registro, inicio de sesión, publicar un viaje, buscar y solicitar un viaje y agregar un conocido.

En este periodo de pruebas, los usuarios experimentarán con las posibilidades que ofrecerá dicha versión, para posteriormente en un formulario comentar todos los aspectos que consideren oportunos con el objetivo de mejorar la aplicación.

9. Implementación

En este apartado se expondrán, por orden cronológico, todos los pasos realizados durante la implementación de la aplicación. Tal y como se especificó en el apartado de Metodología esta fase del proyecto se realizará teniendo en cuenta los conceptos "sprint" y "tarea" de las metodologías ágiles. De este modo, cada una de las diferentes secciones del apartado corresponderá a un sprint, que temporalmente durará una semana, empezando el día 24/05/2021.

En cada uno de los sprints se especificarán las tareas previstas para realizar durante la semana y, además, un resumen de los aspectos más relevantes del desarrollo, así como contratiempos que hayan surgido. Por último, al final de cada sprint se valorará brevemente el trabajo realizado teniendo en cuenta las previsiones y se determinarán las tareas no terminadas para realizarlas en el siguiente sprint.

También siguiendo las pautas de las metodologías ágiles, la implementación de los bloques de la aplicación se realizará en paralelo, por lo que no será estrictamente necesario tener completamente desarrollada una para comenzar con la otra. Sin embargo, por dependencia entre ellas, se priorizarán las tareas teniendo en cuenta el siguiente orden: Base de Datos, API y *Web App*.

Las tareas correspondientes al grupo de Despliegue en Entornos serán las últimas en ejecutarse, pues esperaremos a tener un prototipo funcional bastante avanzado para ejecutar las pruebas y finalmente la aplicación completa sin errores para desplegarla en el entorno de producción.

9.1. Sprint 1: Base de datos y API

Para empezar con el desarrollo de la aplicación, realizaremos en primer lugar la capa de datos, concretamente la Base de datos. Ya que su implementación es la más independiente, todas las tablas se podrán crear directamente a partir del modelo obtenido en la sección de Modelo relacional de la Base de Datos del apartado de 8. Además, el hecho de tenerla completa desde el inicio nos permitirá desarrollar las tareas de la API de forma rápida.

El primer paso realizado ha sido instalar Laragon como entorno de desarrollo, que nos ofrece el administrador de bases de datos HeidiSQL. Tras iniciar el servidor en Laragon, este iniciará dos servicios en dos puertos diferentes. Por una parte Apache en el puerto 80, que no

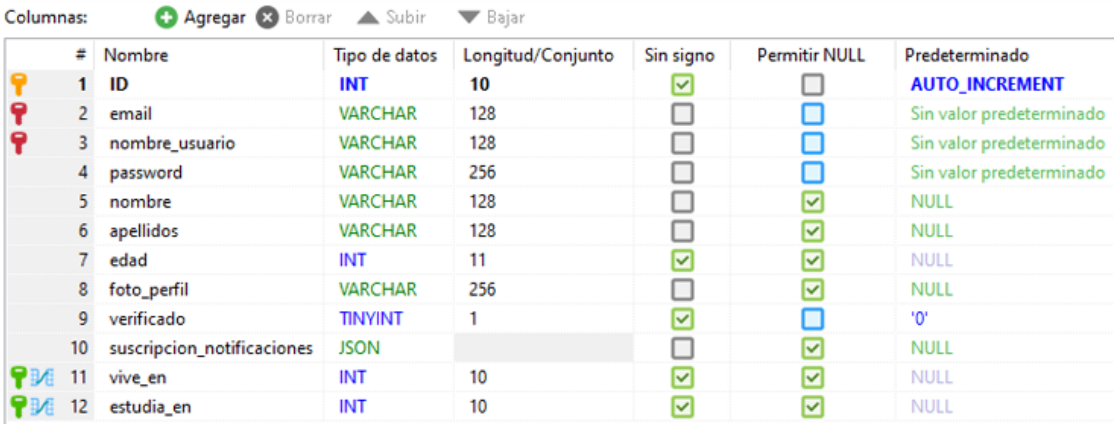


Figura 9.1: Ejemplo de tablas con columnas e índices en HeidiSQL

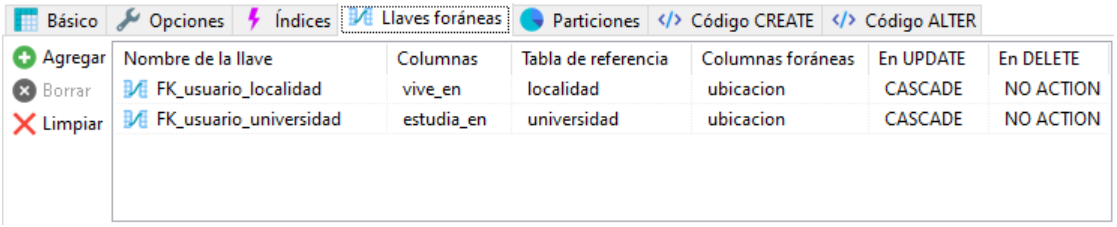


Figura 9.2: Ejemplo de claves ajenas en HeidiSQL
Fuente propia

utilizaremos y por otra MySQL en el puerto 3306, al que conectaremos con HeidiSQL para acceder a la parte administrativa de la base de datos.

Tras crear la base de datos con el nombre "db_unicarapp", siguiendo el diseño de las tablas del modelo relacional mencionado anteriormente se han creado todas las tablas con sus respectivas columnas e índices, además de las relaciones entre ellas por medio de claves ajenas.

En las figuras 9.1 y 9.2 podemos observar respectivamente las columnas con sus índices y las claves ajenas de la tabla usuario. Por lo que respecta a los índices, las columnas que los conforman se indican con el icono con forma de llave. El color amarillo significa clave primaria, el rojo clave alternativa y el verde clave ajena. En cuanto a las claves ajenas, en el editor se especifican las columnas de la tabla junto a la tabla referencia y sus columnas, además de las acciones a realizar en caso de actualización o eliminación en la tabla referenciada.

Finalmente, el resultado obtenido tras crear todas las tablas se puede apreciar en la figura 9.3. De cada tabla se puede ver el tamaño en memoria que ocupa, que actualmente es bajo por tener únicamente datos insertados a modo de prueba. Sin embargo, además de las tablas, también aparecen las consultas almacenadas. Estas se diferencian en la lista por el símbolo y por tener como prefijo en el nombre el verbo de la acción que realizan. Estas consultas han sido implementadas para probar su funcionamiento y, por tanto, no son las definitivas. Más

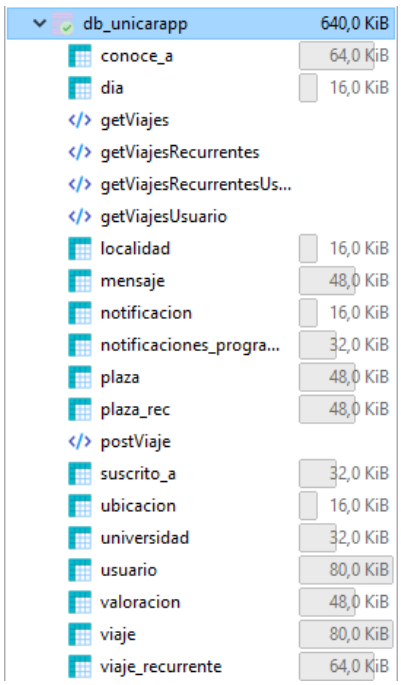


Figura 9.3: Lista de tablas y consultas almacenadas
Fuente propia

adelante en la tarea "Optimización de la DB" se valorará la implementación de más consultas según se considere conveniente.

El objetivo de las consultas almacenadas es tener preparadas consultas que se van a realizar constantemente para que estas se ejecuten de manera más rápida, pues las tiene almacenadas en caché. Además, en la parte del servidor simplifican el código ya que únicamente se debe especificar la función con los parámetros requeridos. Estas son dos ventajas muy atractivas que compensan la principal desventaja que estas consultas ofrecen, como es el uso de memoria en el sistema (*Procedimientos MySQL: ventajas, desventajas y casos de uso - ADN Cloud*, 2019). Este aspecto, sin embargo, hoy en día no es tan relevante ya que el almacenamiento en la nube cada día es más barato.

Un ejemplo de consulta almacenada se puede ver en la figura 9.4, concretamente se trata de la función de búsqueda de viajes. En la parte superior se indican los parámetros de esta, junto a el tipo de datos y el contexto. En la parte inferior se especifica la consulta a realizar, que hace uso de los parámetros. Como contratiempo, ha surgido la duda de cómo tratar los parámetros cuando estos fueran nulos, pues, en la búsqueda, los filtros no se aplican todos. Como solución para cada parámetro perteneciente a los filtros se ha incluido un condicional que devuelve "true" cuando dicho parámetro sea nulo y no afecte así a la consulta. Dichos filtros están incluidos en la cláusula "WHERE" entre las líneas 14 y 22.

Tras finalizar la creación de la base de datos, proceso que se ha realizado de manera sorprendentemente rápida debido a la previa especificación en el apartado de Diseño, el

Opciones | Parámetros | Código CREATE

Agregar | Borrar | Limpiar | Subir | Bajar

#	Nombre	Tipo de datos	Context
1	p_sentido	INT	IN
2	p_fecha	DATE	IN
3	p_fecha_min	DATE	IN
4	p_fecha_max	DATE	IN
5	p_hora_max_ida	TIME	IN
6	p_hora_min_ida	TIME	IN
7	p_hora_max_vuelta	TIME	IN
8	p_hora_min_vuelta	TIME	IN
9	p_origen	INT	IN
10	p_destino	INT	IN

Cuerpo de la rutina:

```
1  BEGIN
2     SELECT
3     v.id, 1*v.sentido sentido, v.fecha, v.hora_salida, v.hora_llegada, org.nombre origen, dest.nombre d
4     (SELECT COUNT(*) FROM plaza AS pla WHERE pla.viaje=v.id AND pla.estado LIKE '1') plazas_ocupadas,
5     (SELECT estado FROM plaza AS pla WHERE pla.viaje=v.id) estado,
6     (SELECT AVG(val.valoracion) FROM valoracion AS val WHERE val.valorado = v.conductor AND val.rol_val
7     FROM viaje AS v
8     inner JOIN usuario AS usu
9        ON usu.id = v.conductor
10    inner JOIN ubicacion AS org
11       ON org.id = v.origen
12    inner JOIN ubicacion AS dest
13       ON dest.id = v.destino
14    WHERE (p_sentido=v.sentido  OR p_sentido IS NULL)
15    AND (p_fecha=v.fecha OR p_fecha IS NULL)
16    AND (p_fecha_min<=v.fecha OR p_fecha_min IS NULL)
17    AND (p_fecha_max>=v.fecha OR p_fecha_max IS NULL)
18    AND (p_hora_max_ida>=v.hora_llegada OR p_hora_max_ida IS NULL)
19    AND (p_hora_min_ida<=v.hora_llegada OR p_hora_min_ida IS NULL)
20    AND (p_hora_min_vuelta<=v.hora_salida OR p_hora_min_vuelta IS NULL)
21    AND (p_hora_max_vuelta>=v.hora_salida OR p_hora_max_vuelta IS NULL)
22    AND (p_destino=v.destino OR p_destino IS NULL);
23 END
```

Figura 9.4: Ejemplo de consulta almacenada
Fuente propia

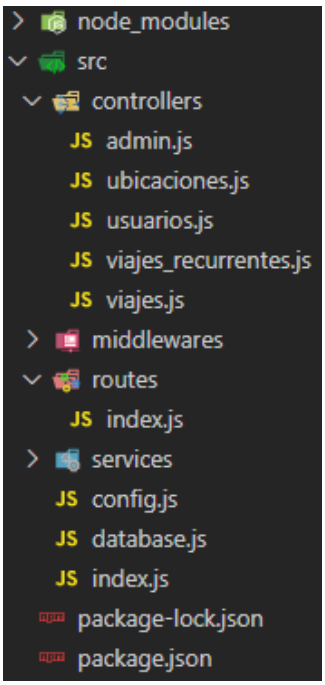


Figura 9.5: Distribución carpetas API
Fuente propia

siguiente paso ha sido empezar con la base de la API. Entendemos como base, la estructura de esta sin especificar concretamente todas las rutas, pero que sirve para tener instaladas las librerías necesarias, distribuir los archivos según su función y configurar la conexión con la base de datos.

Tal y como se especificó en la sección Diseño de la arquitectura tecnológica Front/Back-end de Diseño, se empleará NodeJS junto a Express para implementar la API. Al tratarse de un proyecto de NodeJS independiente a la propia aplicación, este tendrá una distribución de carpetas propia. La estructura se muestra en la figura 9.5

Más allá de los archivos de configuración de cualquier proyecto NodeJS, "package.json" y "packaje-lock.json" y la carpeta "node_modules", en la carpeta "src" se incluye toda la lógica de la API. Esta carpeta, internamente se distribuye de manera sencilla con apenas 4 carpetas y 3 archivos. En este momento de desarrollo, las carpetas "middlewares" y "services" todavía no tienen ninguna función pero se emplearán próximamente. Las otras dos carpetas, "controllers" y "routes", contienen respectivamente los controladores de cada entidad en la API y el archivo con todas las rutas que esta acepta.

En cuanto a los 3 archivos de la carpeta "src", los más relevantes son dos, "database.js" y "index.js", que se pueden observar en los fragmentos de código del anexo A.1 y A.2.

En el primero de ellos se configura el servidor para que este escuche las peticiones en el puerto 3050 y con la ruta raíz "/api". Como aspecto relevante, cabe mencionar la configuración

para que el servidor acepte peticiones desde un origen diferente al propio, ya que la aplicación web se alojará en un proyecto diferente y por tanto tendrá diferente origen.

En el segundo, en cambio, se establece la conexión con la base de datos que se utilizará en los controladores. Para ello, se debe importar del módulo "mysql" y crear una conexión a la base de datos con los parámetros de acceso. Finalmente esta conexión se exporta para poder importarla en los mencionados controladores.

Sin embargo, a pesar de la importancia de estos dos archivos, la lógica de la API recae en los archivos ubicados en las rutas y los controladores.

El fragmento de código del anexo A.3 contiene la estructura del archivo "routes/index.js". Este funciona como índice de todas las peticiones que se pueden realizar en la API. Para implementarlo, en primer lugar se debe crear una instancia del enrutador de Express y de los controladores. La instancia del middleware "router" nos permite añadir de forma muy intuitiva la lista de rutas, de manera que en cada una se indica el verbo de la petición, la ruta y la función del controlador que debe ejecutarse.

En cuanto a los controladores se puede encontrar un ejemplo en el fragmento de código del anexo A.4, concretamente este sería el de "usuarios.js". En cada controlador se importa la conexión a la base de datos mencionada anteriormente y, para evitar conflictos con la sincronía propia de JavaScript, dicha conexión se pasa por la función "promisify" del módulo "util" de NodeJS, que nos permite realizar las consultas de forma asíncrona. Esto resulta necesario en casos en los que en una función se realicen varias consultas dependientes entre ellas, es decir, que a partir de los resultados de una se realice la siguiente. El resto de contenido del controlador son las propias funciones para ejecutar las consultas, que finalmente se exportan para poder ser utilizadas en el enrutador.

Un ejemplo de función con una consulta a la base de datos es el fragmento del anexo A.5. En cada función, en primer lugar se recogen los datos de los parámetros para formar así la consulta a realizar. A continuación, se realiza la consulta a la base de datos de forma asíncrona para garantizar que se envíe la respuesta antes de haber obtenido el resultado. Todo ello se realiza dentro del bloque de instrucciones try...catch, que nos permite enviar un resultado de error en caso de producirse una excepción durante la consulta o la ejecución de la función. En caso de éxito la respuesta se enviará con el estado 200 que corresponda y en caso de error, al tratarse de un error de servidor, devolverá un estado 500. Todo ello, para poder tratarlo fácilmente en la parte del cliente se enviará en formato json.

Finalmente, a partir de la estructura del controlador y la documentación de la API del apartado de Diseño, donde se especifican todas las rutas, se han desarrollado todas las consultas correspondientes a las entidades viajes y usuarios. Esto nos permitirá en el siguiente sprint empezar con la base de la aplicación web e implementar las primeras interfaces ya con los datos servidos desde la API.

Así pues, el trabajo realizado durante esta primera semana ha resultado productivo ya que

una vez desarrollada la base de la API y teniendo gran parte de las rutas, añadir las entidades restantes con sus rutas no requiere demasiado tiempo y se hará a medida que se tengan las interfaces de la aplicación más avanzadas.

9.2. Sprint 2: Base de la *Web App* y primeras interfaces

Una vez implementadas las rutas de la API para la gestión de viajes y usuarios, el siguiente objetivo es crear la base de la aplicación para, a continuación, comenzar con el desarrollo de las interfaces.

Tal y como se especificó en la sección de Diseño de la arquitectura tecnológica Front/Back-end, para la aplicación se creará un proyecto independiente, aunque también será desarrollado en el entorno NodeJS. La distribución de las carpetas es similar y, en este caso, se incluye la carpeta "public" que contendrá todo el contenido a servir en el lado del cliente.

En los archivos de configuración del proyecto se indica que todas las peticiones al servidor se realicen a partir de dicha ruta, por lo que el resto de archivos quedan inaccesibles desde la parte del cliente. Además, al tratarse de una aplicación de una única página, cuando se acceda a cualquier ruta desde el navegador, el servidor enviará siempre la plantilla del índice, en el que ya la librería AngularJS incluirá la vista correspondiente a la ruta.

Ya con el servidor configurado, el próximo paso ha sido realizar la distribución de las carpetas y archivos en esta carpeta pública, atendiendo al patrón Modelo-Vista-Controlador que permite aplicar AngularJS y facilita la refactorización del código. Esta distribución puede observarse en la figura 9.6.

En este momento del desarrollo, la raíz de esta carpeta únicamente contiene los archivos "app.js" y "index.html" y las carpetas en las que se distribuirán el código según su función. Inicialmente las únicas con contenido son la carpeta con los estilos y las 3 básicas para AngularJS: "controllers", "services" y "views".

Del archivo "index.html", que servirá el servidor al acceder a cualquier ruta desde navegador y cuya primera versión se puede ver en el fragmento de código del anexo A.6, destaca un elemento div con el atributo ng-view sobre el que Angular insertará la vista que se le especifique según la ruta. Además, a continuación de este, se incluyen todos los scripts necesarios. En primer lugar serán importadas las librerías de terceros, como AngularJS y sus extensiones, seguidas de los servicios, que gestionarán las peticiones a la API, y los controladores de todas las vistas. Finalmente, el último script importado es el que contiene el módulo de la aplicación, cuya primera versión está en el fragmento de código del anexo A.7

Este archivo "app.js" es fundamental para el funcionamiento de la aplicación. En él se define el módulo principal de la aplicación con AngularJS y, además, reunirá la importación del resto de módulos que empleará la aplicación, desarrollados próximamente. También se

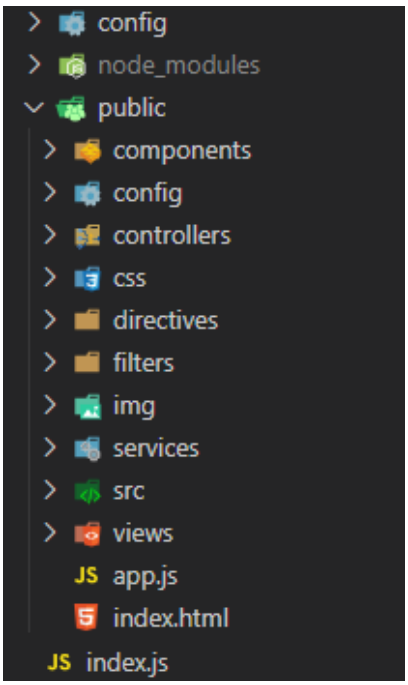


Figura 9.6: Distribución de carpetas de la aplicación web
Fuente propia

incluye en este archivo la lista de rutas aceptadas por la aplicación, indicando para cada una la vista a mostrar y el controlador correspondiente.

Después de haber preparado la base de la aplicación, el siguiente paso ha sido empezar a desarrollar la primera interfaz funcional. Pero antes de ello, para trabajar sobre datos servidos desde la API, se ha implementado el servicio correspondiente a la entidad "viaje" de la API. Como se ha mencionado anteriormente, en los servicios se realizan todas las peticiones de manera que estas queden todas juntas según la entidad a la que se haga la petición. Concretamente, para el proyecto se han empleado factorías, que funcionan como contenedores de código para poderlo reutilizar en los diferentes controladores.

En este proyecto encontramos 3 entidades principales, por lo que se implementarán tres servicios: "Usuarios, "Viajes recurrentes" y "Viajes". De esta última encontramos, en el fragmento de código del anexo A.8, la estructura del servicio, que es igual a las otras dos. Cada una de las peticiones se incluye en una función y esta se exporta en el servicio para poder ser utilizada en cualquier controlador.

Llegado este punto, ya es posible crear la primera interfaz con datos recogidos desde la API y accesible desde el enrutador de AngularJS. Para ello, únicamente será necesario crear dos archivos por interfaz: la vista y el controlador, que se ubicarán en sus respectivas carpetas.

Por lo que respecta a la vista, en formato html, esta incluirá los elementos que se visualizarán. Sin embargo, como aspecto relevante por trabajar con la librería AngularJS, esta nos

Figura 9.7: Interfaz "Mis viajes"
Fuente propia

permite incluir cierta lógica en esta plantilla html por medio de atributos en los elementos. De esta manera, el manejo de los elementos del DOM no recae completamente en el código JavaScript.

También por trabajar con esta librería es posible compartir las variables entre la vista y el controlador. Esto ser realiza mediante el objeto $scope, que permite acceder a las variables definidas en el controlador desde la vista, actualizándose su valor de forma automática cuando este cambia.

Un ejemplo del manejo del DOM mediante atributos en la plantilla html podría ser la ocultación de un botón según se cumpla o no una condición. En el propio botón se incluiría el atributo ng-If=".ondicion" y este elemento se ocultaría o se mostraría en función de una condición contenida en el controlador.

Otro atributo realmente interesante que permite añadir AngularJS es ng-repeat=".lemento in array". Este actúa en bucle incluyendo el contenido del elemento en cuestión tantas veces como posiciones tenga el array indicado. Además, se puede acceder a cada elemento del array para mostrar sus valores en la plantilla. Un ejemplo de uso de esta funcionalidad es la lista de viajes de la interfaz de "Mis viajes" en la figura 9.7

Figura 9.8: Ejemplos de tarjetas de viaje
Fuente propia

Con respecto al diseño de esta interfaz incluido en los wireframes, se ha modificado la tarjeta de viaje, que sin desplegar únicamente muestra el sentido, la hora y el rol. Esto se ha hecho para minimizar el espacio que ocupa y, además, porque el resto de información no es tan relevante, pudiendo mostrarse al desplegar la tarjeta. En la figura 9.8 se exponen diferentes posibilidades de dichas tarjetas según el rol y el estado del viaje.

Otro cambio respecto al diseño ha sido el selector de la semana. Inicialmente se trataba de un desplegable que incluía todas las semanas del curso actual, desde la primera semana de septiembre hasta la última de agosto, mostrando como semana seleccionada la actual. Sin embargo, este selector no resultaba muy usable puesto que la cantidad de opciones era muy elevada. Como alternativa, se ha empleado una especie de paginación que muestra la semana seleccionada y dos flechas para navegar hacia la siguiente semana o la anterior. Si bien con el selector resultaba igual de sencillo navegar a cualquier semana del año, este cambio simplifica la navegación a la semana anterior y posterior que, junto a la actual, serán de mayor interés en comparación a las demás.

La finalización de esta interfaz supone el final de este sprint en el que el trabajo realizado supera las expectativas previstas. Tras un pausado inicio asimilando los conceptos relacionados con la librería AngularJS, la base desarrollada con la distribución de los archivos y un primer ejemplo de interfaz sirven de guía para realizar las demás interfaces del mismo modo. Esto supone un gran avance puesto que los siguientes, en cuanto al desarrollo, son repetitivos y por tanto sencillos de realizar.

9.3. Sprint 3: Control de usuarios e implementación de interfaces

En este tercer sprint, tal y como indica su título, los dos objetivos principales son, por una parte, implementar el inicio de sesión y el registro de usuarios y, por otra, continuar con las demás interfaces.

Volviendo al proyecto de la API, para el control de usuarios será necesario crear las rutas para el inicio de sesión y el registro. En este momento, surge la necesidad de tratar el codificado de la contraseña del usuario para no guardarla directamente en la base de datos.

Para ello, en el controlador de la entidad usuario importaremos la librería "bcrypt", que nos servirá para codificar la contraseña en el proceso de registro. Antes de guardar el registro del usuario en la base de datos, se genera un código de 10 caracteres y, a continuación, con este se codifica la contraseña y se devuelve el resultado para finalizar el registro. Y ya con el usuario registrado, para iniciar sesión, también con esta librería se compara la contraseña insertada con la del usuario que intenta realizar el inicio de sesión. En el fragmento de código del anexo A.9 se ven ambos procedimientos implementados con la librería bcrypt.

Cabe mencionar que para verificar el perfil del usuario se ha empleado un sistema de verificación mediante email. De esta manera, en el momento en que se realice el registro se envía un correo electrónico al usuario, cuyo contenido se puede ver en la figura 9.9. Tras visitar el enlace proporcionado, el usuario será verificado y se le direccionará a la página de inicio. Para este procedimiento de envío de correo electrónico se ha empleado el módulo "nodemailer".

También relativo al control de usuarios, resulta necesaria la implementación de un sistema de autorización y autenticación para el acceso a la API. Como se especificó en el Diseño, este procedimiento se realizará mediante token. La librería empleada en este caso será "jwt-simple" que nos permite crear y decodificar los tokens que contienen información del usuario al que pertenecen.

En este momento, pasan a ser utilizadas dos carpetas del proyecto de la API mostradas en la figura 9.5, concretamente "services" y "middleware". La primera de ellas contendrá funciones para la codificación y decodificación del token. Para la creación del token este podrá ser de 2 tipos. Por una parte, el token incluido en la url del correo de verificación y, por otra parte, el token que permitirá el acceso a la API. Este token contendrá el identificador del usuario, la fecha en que se crea, la fecha de expiración y una variable booleana que indicará si el token pertenece a un usuario administrador o no. La fecha de expiración será en el caso del primer tipo de token 1 hora después de ser creado y en el caso del segundo 14 días.

En cuanto a la carpeta "middleware", esta contendrá fragmentos de código que, gracias a la naturaleza del entorno NodeJS, se emplearán como procedimientos intermedios durante el manejo de las peticiones en el enrutador. Esto nos permitirá en cada una de las rutas especificadas, incluir la función que verifique la validez del token para continuar con la petición

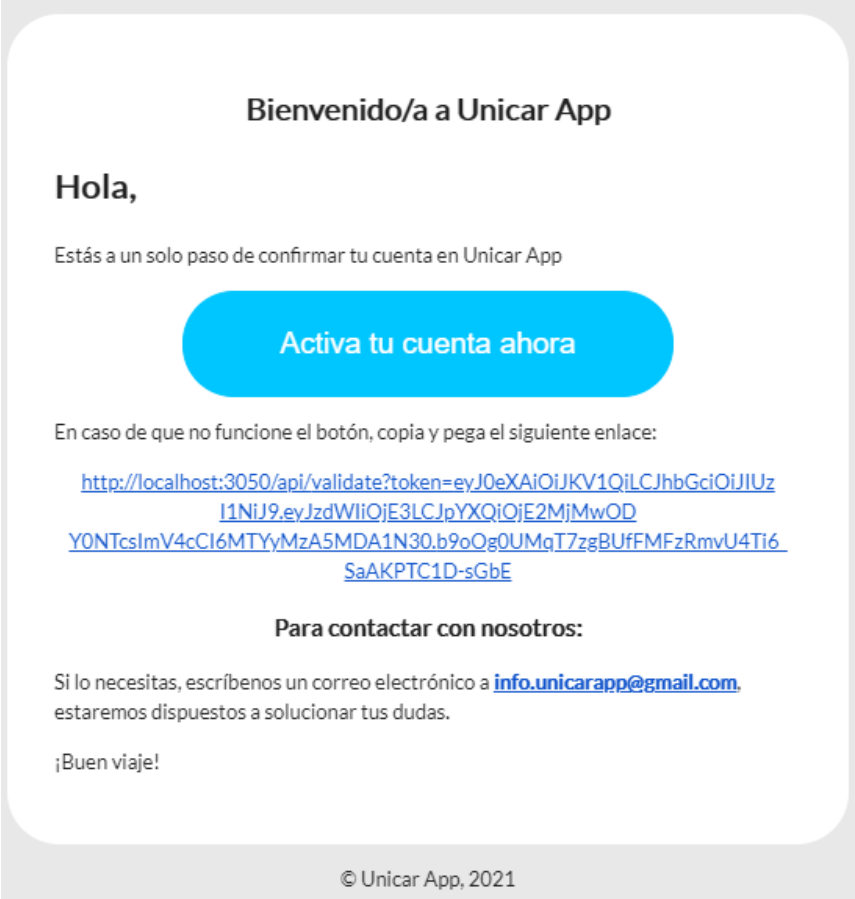


Figura 9.9: Email de verificación de usuario
Fuente propia

en caso de ser válido o enviar una respuesta con estado 403 por no estar autorizado, bien por no ser un token válido o caducado.

Las principales partes de este procedimiento relativo a los tokens están incluidas en el fragmento de código del anexo A.10

Ya con el control de usuarios implementado en la parte de la API, pasamos de nuevo al proyecto de la aplicación web para desarrollar las interfaces tanto de registro como de inicio de sesión. El resultado obtenido está en la figura 9.10. Ambas son dos interfaces muy simples que únicamente contienen un formulario con los campos necesarios. Para el inicio de sesión estos campos son el email y la contraseña, a los que se les añade un tercero como es el nombre de usuario en el caso del registro. Los otros dos elementos de las interfaces son los botones que realizan la acción principal y un texto que enlaza ambas interfaces. De esta manera, los usuarios no registrados podrán acceder al registro y los registrados al inicio de sesión.

Como aspecto relevante al iniciar sesión, el servidor devolverá un token generado junto al identificador del usuario, que se almacenarán en el "localStorage" del navegador. De esta manera, el usuario no deberá iniciar sesión cada vez que entre en la aplicación. Sin embargo, el token caduca cada 14 días, por lo que si no se vuelve a iniciar sesión pasado este periodo, el servidor devolverá una respuesta notificando que el token ha caducado. En este caso se requerirá de nuevo iniciar la sesión, renovándose así el token.

Una vez el usuario haya iniciado sesión, este será dirigido a la interfaz de "Mis viajes". En esta interfaz, expuesta en el sprint anterior, se visualizarán los viajes de la semana actual, y a través de los botones de cada viaje se podrá acceder a diferentes interfaces.

En esta interfaz, los viajes serán diferenciados entre futuros y pasados, los futuros en caso de ser conductor tendrán la posibilidad de ser editados o borrados y de acceder al chat o las solicitudes. En este momento de desarrollo se ha realizado la interfaz de solicitudes, visible en la figura 9.11. En esta se incluye la información del viaje y la lista de solicitudes, con la posibilidad de aceptarlas o rechazarlas.

En cuanto a los viajes pasados, bien sean como conductor o como pasajero, se podrá acceder a valorar a los usuarios que han participado en él. En esta interfaz, al igual que en la de solicitudes se visualizará la información del viaje y, además, la lista de usuarios, con el emoticono relativo al rol que han tenido en el viaje y sobre el que se realizará la puntuación. Para introducir dicha valoración, se deberá pulsar sobre las estrellas y finalmente pulsar el botón de valorar para enviarla.

Para finalizar con las interfaces relativas a los viajes, las restantes son la de publicación y la de búsqueda de estos. Como podemos apreciar en la figura 9.12, ambas comparten la misma estructura, con el formulario y el botón que realiza la acción de buscar o publicar. En el de publicar, además, se incluye la opción introducir el número de plazas disponibles. Cabe decir que la opción de añadir pasos intermedios y pasajeros asignados todavía no han sido implementadas.

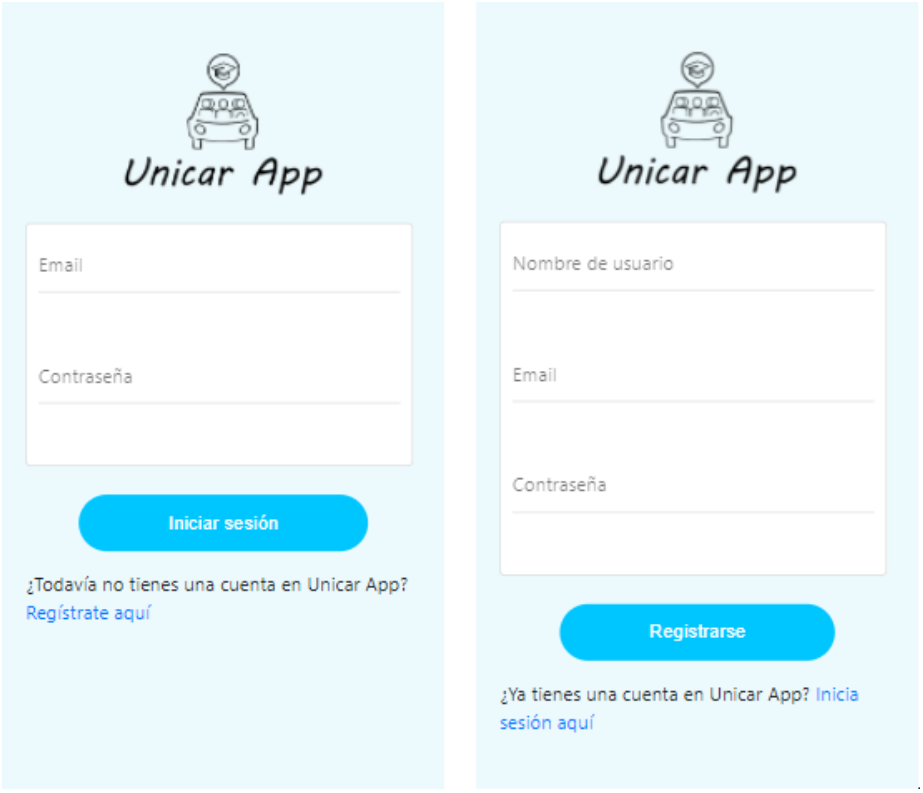


Figura 9.10: Interfaces “Inicio de sesión” y “Registro”
. *Fuente propia*

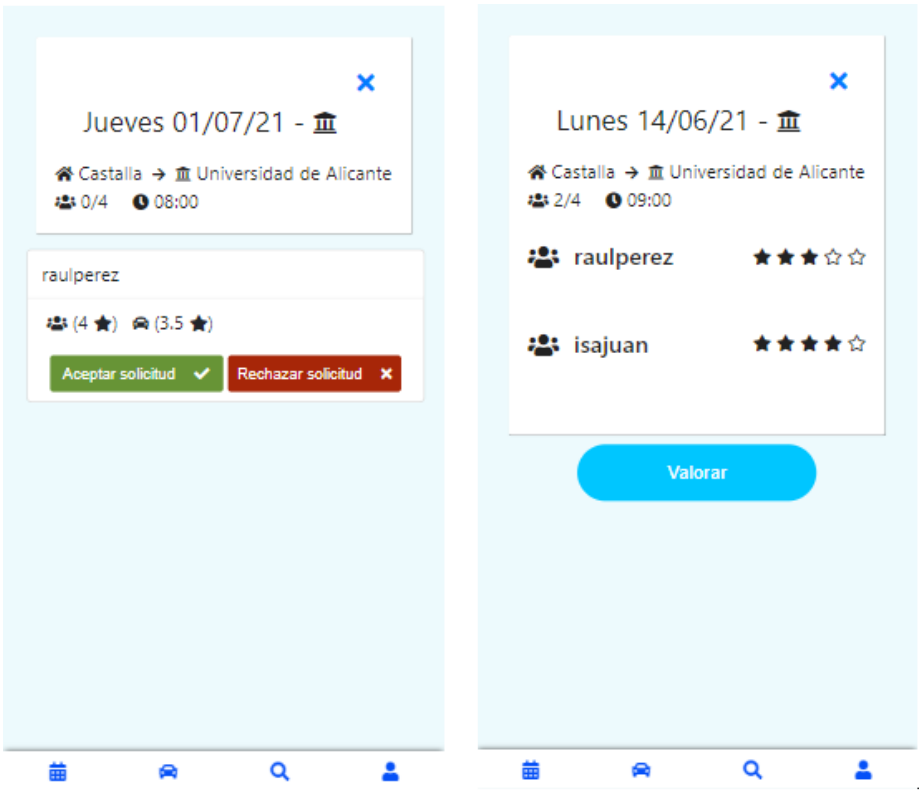


Figura 9.11: Interfaces "Solicitudes" y "Valorar"
. *Fuente propia*

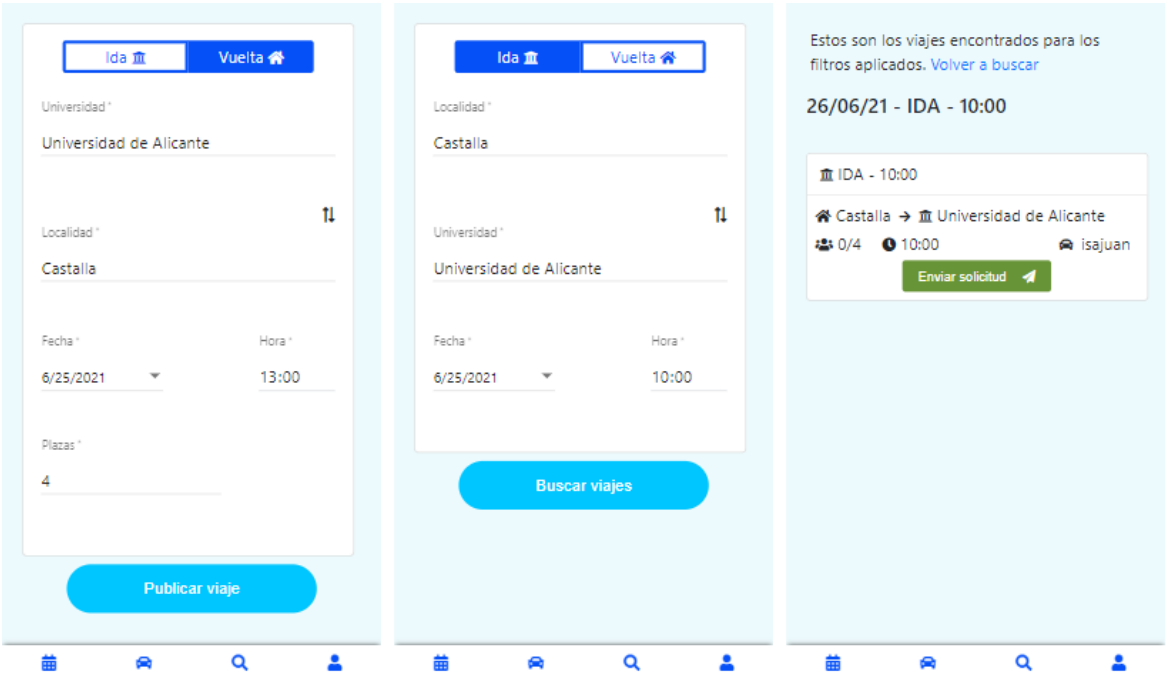


Figura 9.12: Interfaces "Publicar viaje", "Buscar viaje" y "Viajes encontrados" . *Fuente propia*

Ambos formularios, con el objetivo de mejorar la usabilidad en esta interfaz, tendrán por defecto los datos del usuario para la universidad y localidad, y la fecha y hora actual. Además, en caso de que el usuario necesite cambiar el origen o el destino, a medida que se escriba en estos campos, aparecerán ubicaciones sugeridas para autocompletar. Este código, complejo por tener los ajustes del autocompletado, ha sido implementado en un controlador aparte, que simplemente se debe importar en la vista a utilizar.

La interfaz de los viajes encontrados en la búsqueda es similar a la de mis viajes con varios aspectos que las diferencian. Principalmente desaparecen los botones de la parte superior, además de los días de la semana. Este espacio superior lo ocupará el enlace para volver a buscar y los filtros introducidos. En el espacio restante, por ser más grande y para poder visualizar directamente la información de los viajes, las tarjetas estarán desplegadas. De esta manera a simple vista se podrán comparar los resultados obtenidos. Cabe destacar que en la búsqueda, la hora introducida tendrá cierto margen para encontrar viajes con hora similar. Actualmente este margen es de 30 minutos aunque posteriormente se introducirá la posibilidad de ajustarlo en el formulario.

Una vez ya implementada la gestión de los viajes, es momento de completar las interfaces correspondientes al perfil del usuario. Estas dos interfaces son la del perfil y la de editar perfil, que se pueden encontrar en la figura 9.13.

En la página del perfil, que muestra los datos del usuario, se ha realizado un cambio respecto al diseño inicial de las interfaces. Los ajustes de las notificaciones, que estaban en la vista de

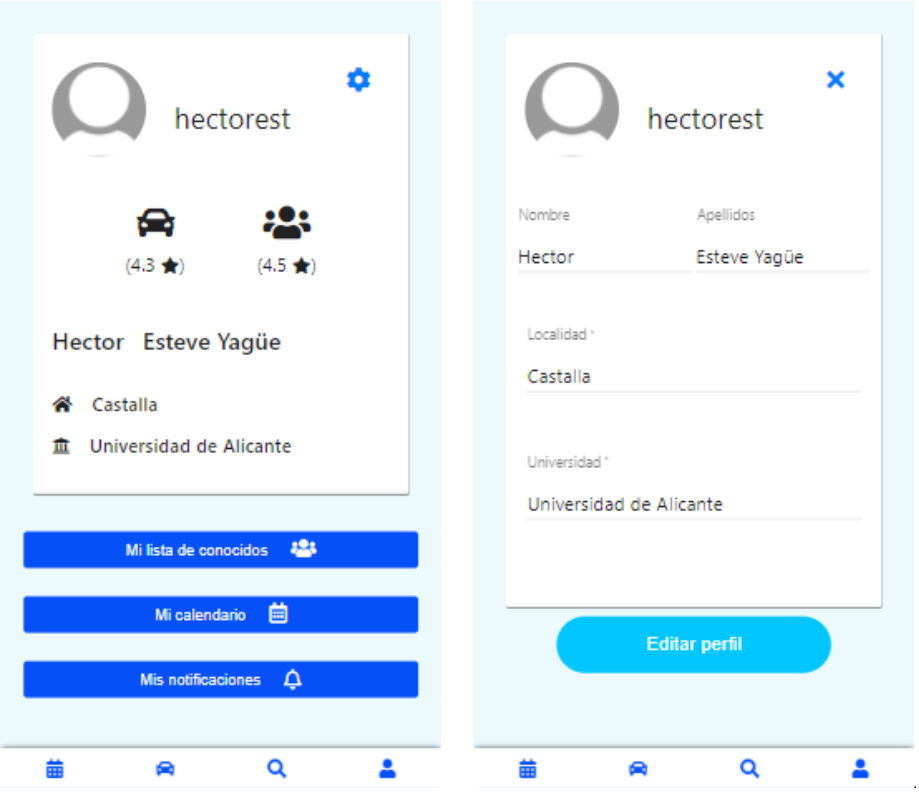


Figura 9.13: Interfaces "Mi perfil" y "Editar perfil".
Fuente propia

editar el perfil, ahora han pasado a incluirse en una página aparte, accesible desde el perfil. Las otras dos páginas accesibles mediante los botones inferiores son la lista de conocidos y el calendario.

La vista de editar perfil muestra por defecto los datos del usuario en el formulario, pudiendo ser editados. Los campos de localidad y universidad son manejados desde el controlador propio implementado anteriormente, lo que ha supuesto un ahorro considerable de tiempo.

Ya finalizando con el sprint actual, las interfaces a implementar son las que se pueden acceder desde la página de perfil: la lista de conocidos y el calendario.

Con respecto a la página de los conocidos, de la figura 9.14, el comportamiento de esta incluye dos interfaces. Por una parte, la vista genérica, que incluye las solicitudes recibidas por parte de usuarios y la propia lista de conocidos. Al igual que los viajes, los elementos de las listas son representados mediante tarjetas, que al desplegarse permiten visualizar las valoraciones del usuario en cuestión y los botones para gestionar las solicitudes o eliminar el usuario de la lista.

En la parte superior de esta página está situado un campo de texto para buscar un usuario a partir de su nombre de usuario. Este campo, en el momento en que se han introducido 3

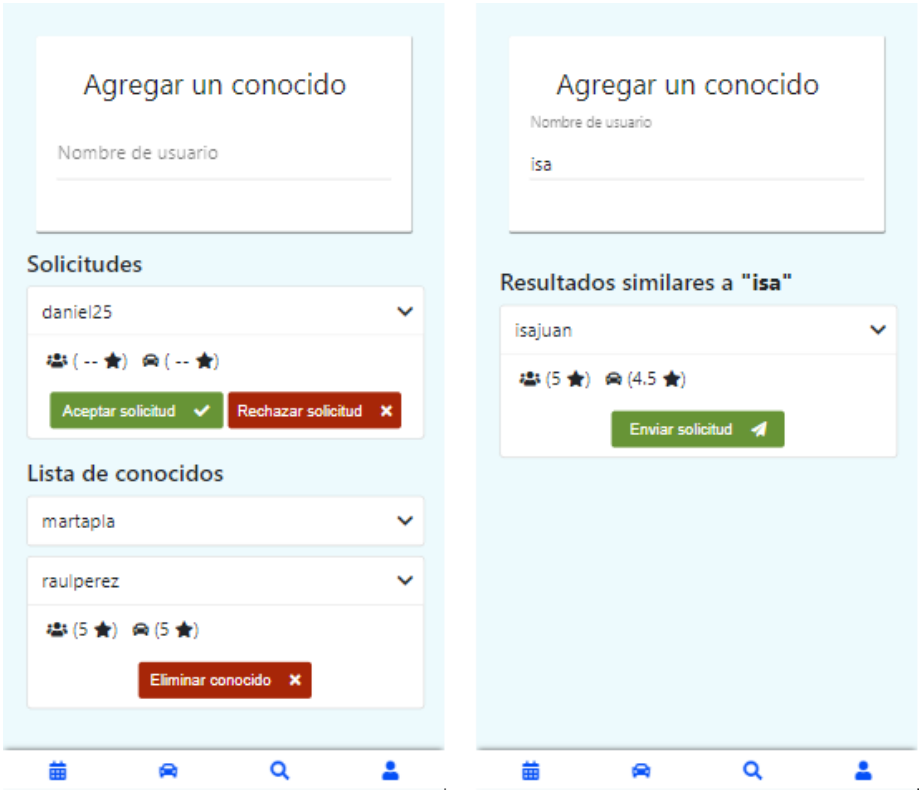


Figura 9.14: Interfaces "Mi lista de conocidos" y "Búsqueda de conocidos". *Fuente propia*

caracteres muestra en la parte inferior los usuarios cuyo nombre de usuario coincide en parte con la cadena introducida. Los resultados de esta búsqueda se actualizan a medida que se cambia el texto introducido, siempre que el número de caracteres sea 3 o superior.

La última interfaz realizada en este sprint se trata del calendario. Para ello, se tomará como referencia la interfaz de "Mis viajes" con la que comparte muchos aspectos. En primer lugar, esta vista también está paginada por semanas, mostrando en primera instancia la semana actual. Si el usuario no ha introducido su horario de la semana, todos los días aparecerán como días sin clase, junto a un enlace para editar dicho día. En cambio, los días en que el usuario tenga establecidas su hora de entrada y salida estas se mostrarán con la posibilidad de ser editadas. Estas dos vistas se pueden ver en la figura 9.15.

Como conclusión de este sprint, teniendo en cuenta la gran cantidad de interfaces implementadas este ha vuelto a resultar productivo. Sin embargo, en varias interfaces quedan aspectos por pulir que serán corregidos más adelante, sobretodo durante la etapa de pruebas y validación.

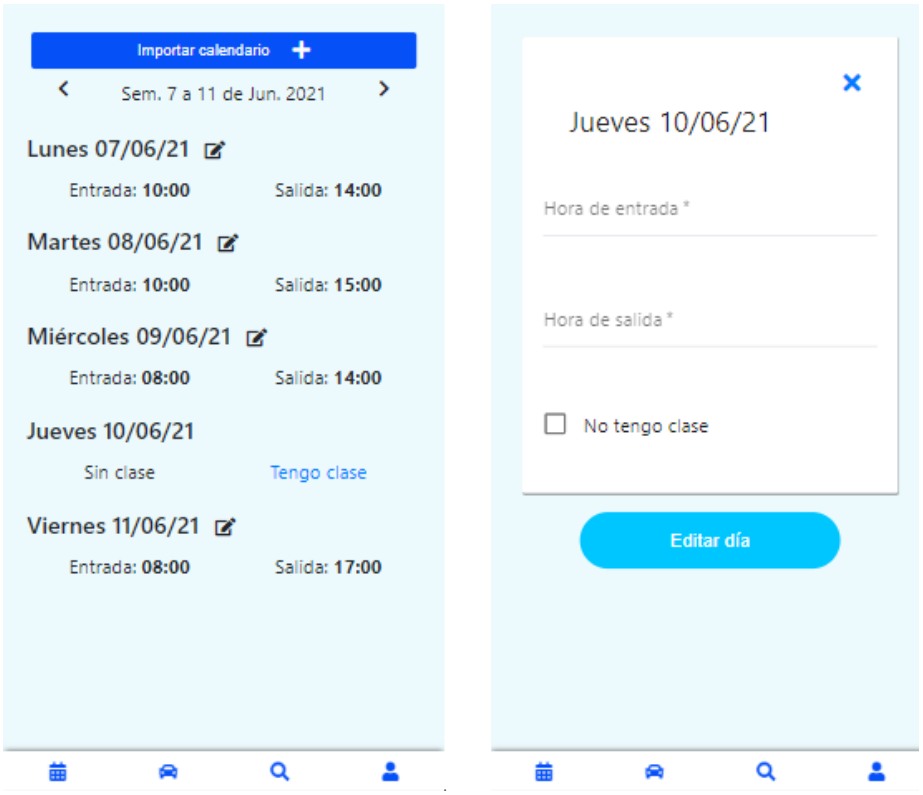


Figura 9.15: Interfaces "Mi calendario" y "Editar día".
Fuente propia

9.4. Sprint 4: Integración PWA y Chat

El cuarto sprint tiene como objetivos integrar dos funcionalidades que, más allá de la implementación de interfaces, requieren más técnica en cuanto a las tecnologías a emplear. Se trata de la integración de la PWA (Progresive *Web App*) y del chat de los viajes.

Por lo que respecta a la PWA, en primer lugar es necesario tener en cuenta cuáles son los requisitos que debe cumplir la aplicación para que sea considerada como una Aplicación Web Progresiva por el navegador Google Chrome. Toda la implementación se ha basado en dicho navegador por ser el más extendido entre los usuarios.

El primer requisito a tener en cuenta se trata de una conexión segura garantizada por el protocolo HTTPS. Sin embargo, durante la implementación este requisito no es necesario, desplegando la aplicación en Localhost, es decir, en un servidor local. Más adelante se desplegará en el entorno de pruebas de Heroku, el cual establece la conexión de forma segura y permite probar esta funcionalidad.

Además, se debe proporcionar un archivo en formato JSON conocido como Manifest, que incluye las propiedades básicas de la aplicación, además de instrucciones para orientar al navegador sobre cómo mostrar la aplicación. Este archivo se debe incluir en la carpeta raíz, junto al "index.html". Para enlazar ambos archivos, en este último se debe incluir la siguiente etiqueta en la cabecera: <link rel=".anifest" href=".anifest.json".. También relacionado con este requisito, es recomendable de cara a orientar a los navegadores incluir las cabeceras incluidas en el fragmento de código del anexo A.11.

El último requisito se trata del *service worker*, descrito detalladamente en el apartado de Estado del arte. Explicado de manera muy breve, este es el responsable de almacenar en caché todos los archivos, notificaciones push, actualización de contenido, manipulación de datos, entre otras funciones. Así pues, se trata de un script que trabaja en segundo plano que intercepta las peticiones entre servidor y cliente para garantizar respuestas válidas incluso sin conexión, ofreciendo datos almacenados en el caché.

Como aspectos fundamentales en la implementación del *service worker*, destacan los captadores de eventos que se incluyen en este. Estos eventos pueden ser peticiones al servidor, instalación de la PWA o incluso notificaciones de tipo push, que se implementarán más adelante.

El captador más interesante es el de peticiones, conocido como 'fetch'. Una vez captado dicho evento, se distinguen varios tipos de peticiones, entre los que destaca, en este momento, el de tipo navegación. Este se produce cuando un usuario accede a una página o navega entre ellas. Captar este evento es esencial para que la aplicación sea accesible sin conexión, puesto que una vez no haya respuesta del servidor, el *service worker* será capaz de ofrecer una página almacenada en caché.

Figura 9.16: Página de ejemplo sin conexión

En este momento de desarrollo, la optimización de esta parte del *service worker* no es primordial, por lo que la respuesta a una pérdida de conexión se solventará ofreciendo como resultado una página que notifique que el dispositivo se encuentra sin conexión. Esta página se puede visualizar en la figura 9.16

Una vez desarrollados el *service worker* y el Manifest, el siguiente paso determinante para la PWA se trata de la instalación en dispositivos. Para ello, durante la implementación, se han realizado las pruebas con un dispositivo Android y con el navegador Google Chrome, los cuales soportan esta característica de "Añadir a pantalla de inicio", tal y como se especifica en el apartado de Estado del arte.

Cabe mencionar que, de cara a la experiencia de usuario, para recrear más fielmente el proceso de descargar una aplicación, se ha creado una vista provisional en la aplicación que contiene un único botón con el título descargar. Cuando este botón es pulsado, empezará el proceso de instalación.

De esta implementación, destaca que el comportamiento por defecto del navegador chrome al detectar que la web se trata de una PWA, este ofrece al usuario mediante un mensaje modal la posibilidad de añadirla a la pantalla de inicio. Sin embargo, mediante un script, este evento ha sido detenido y almacenado para que sea el usuario que lo vuelva a reactivar

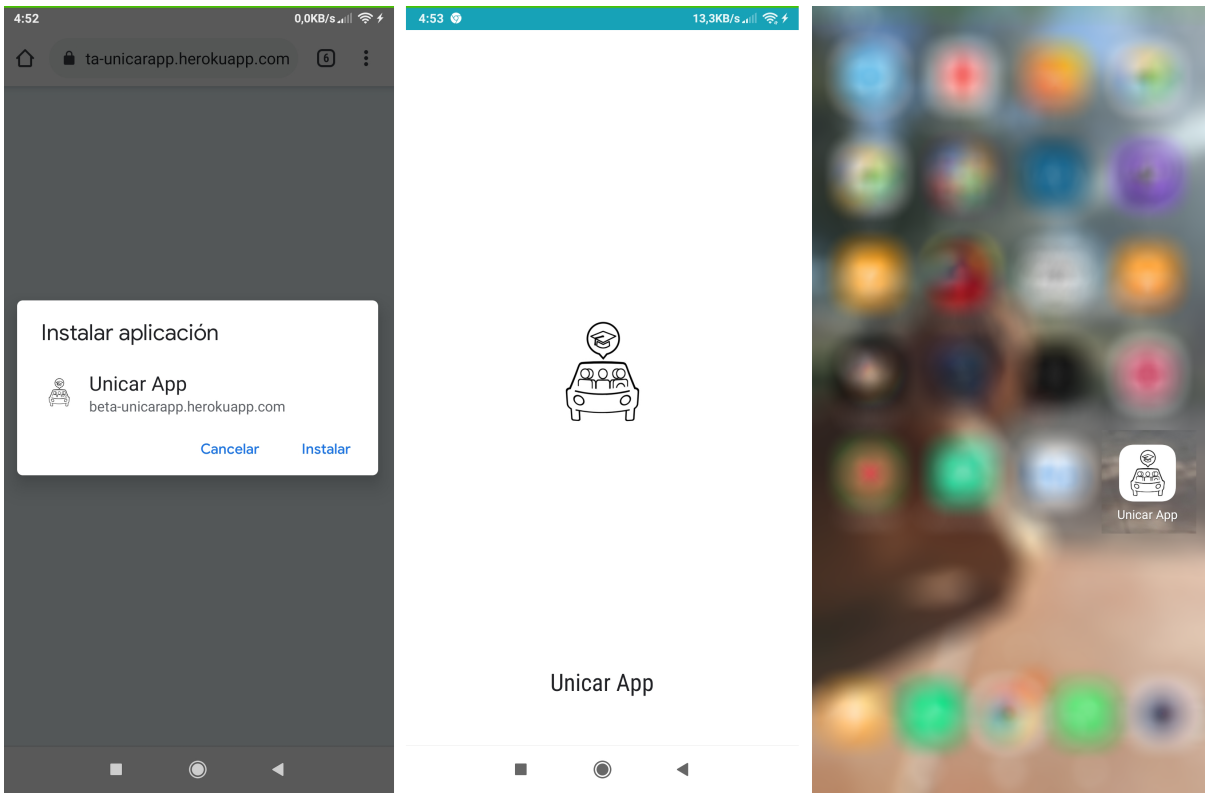


Figura 9.17: Capturas de pantalla PWA instalable
Fuente propia

al pulsar el mencionado botón.

En la figura 9.17 están ordenadas de derecha a izquierda tres capturas relativas a la instalación. La primera de ellas se trata del modal que aparecerá al pulsar el botón de descarga. A continuación, en caso de instalarla, la apariencia de esta será idéntica a una aplicación nativa, con su página de carga con el logotipo (especificado en el Manifest) y el acceso en la pantalla de inicio.

Continuando este sprint con el segundo objetivo propuesto, este se trata del chat. Este tiene la particularidad de que la actualización en esta interfaz debe ser estrictamente en tiempo real para evitar una experiencia de usuario nefasta en caso de tener que recargar la página para ver los mensajes recibidos. La actualización en tiempo real se puede implementar mediante la librería socket.io.

Para empezar con la configuración de esta librería es necesario, en primer lugar, solicitar las claves pública y privada de la aplicación que autorizarán la transferencia de datos entre cliente y servidor por medio de un socket, es decir, un mecanismo de intercambio de paquetes entre dos direcciones IP. La llave pública será la que se introducirá en el código de la parte del cliente mientras que la privada se mantendrá oculta en el servidor.

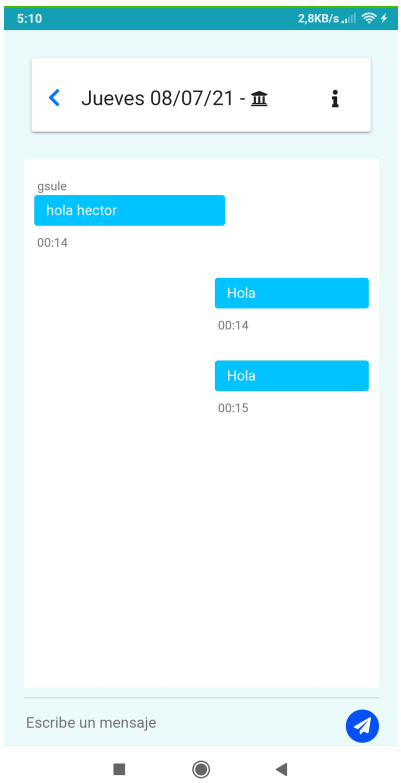


Figura 9.18: Interfaz de Chat de viaje
Fuente propia

El funcionamiento de esta librería se basa, a su vez, en dos librerías diferentes, una para la parte del cliente y otra para la parte del servidor. Sin embargo, la conexión entre ellas se realizará mediante eventos que contendrán en su configuración la llave generada.

Para tener un control de los usuarios que están conectados a algún chat, en la parte del servidor se dispondrá de una lista de chats activos que contendrá en cada elemento, a su vez, una lista de usuarios conectados por cada chat. De estos usuarios se almacenará su identificador y la suscripción a los eventos que genera la librería socket.io y que permite intercambiar eventos con el servidor.

Así pues, tanto en la parte del cliente como en la del servidor habrá dos acciones clave: emitir y escuchar eventos. Cuando un usuario introduce un mensaje en el chat, se debe emitir un evento de nuevo mensaje, que el servidor estará esperando y en ese momento, el servidor emitirá un evento para los usuarios conectados en ese momento en el chat, que enviará el mensaje. Por cada evento de recepción de mensaje, se actualizará la lista de mensajes de la parte del cliente.

La figura 9.18 representa una captura de pantalla de la interfaz correspondiente al chat, en la que aparecen los mensajes intercambiados entre los usuarios del viaje.

Como conclusión tras terminar este sprint, el trabajo ha resultado más dificultoso respecto a los anteriores, pues tanto en la implementación de la PWA como en el Chat, había conceptos nunca tratados y que por tanto era necesario detenerse para entender el funcionamiento de estos. Por esta razón, ambas funcionalidades no están completamente pulidas al término del sprint, lo que supondrá, en un futuro, continuar con estas tareas.

9.5. Sprint 5: Notificaciones Push y Despliegue en Entornos

Este quinto sprint, al igual que el cuarto, conllevará un tiempo extra dedicado a la investigación sobre el uso de librerías y tecnologías. Por tanto, vuelven a ser tareas más técnicas que las relacionadas con las interfaces y que pueden complicarse durante el desarrollo.

El primer objetivo de este sprint es implementar el envío de Notificaciones Push. La librería que nos permitirá realizar esta acción es web-push. En este momento vuelve a aparecer el papel del *service worker*. En este caso para implementar un captador de eventos para las notificaciones: 'push'. Este captador en la parte del cliente, tras recibir la información por parte del servidor, la mostrará en forma de notificación. Sin embargo, previo a todo ese proceso, en la parte del servidor se debe haber emitido este evento a captar con un usuario específico.

Lo primero a tener en cuenta es que para recibir una notificación debe haber un destinatario concreto. Este destinatario, de cara a la librería web-push, que será la que enviará la notificación, se trata de una suscripción. Esta suscripción es la almacenada por cada usuario en la base de datos en caso aceptar el envío de notificaciones a su dispositivo.

Así pues, para generar dicha suscripción en Google Chrome (navegador utilizado durante el desarrollo) se deberá registrar en el gestor de notificaciones del navegador una clave pública generada con la librería. De este registro, que deberá ser permitido por el usuario mediante un mensaje modal lanzado por el navegador, se obendrá la suscripción a almacenar.

Una vez almacenada la suscripción, en formato JSON, en la base de datos, para enviar una notificación a un usuario bastará con consultar la suscripción y añadirla a las opciones de envío de notificaciones, junto a la información a mostrar.

En la figura 9.19 se incluye un ejemplo de cómo funcionan las notificaciones. En primer lugar el usuario deberá permitir las notificaciones en el primer checkbox. En este momento se iniciará el registro en el gestor de notificaciones, que deberá aprobar el usuario con el mensaje modal de la segunda captura. Finalmente, la tercera captura se trata de una notificación recibida, concretamente tras recibir una solicitud de viaje.

Llegado este momento de desarrollo, la aplicación ya tiene gran parte de sus funcionalidades implementadas, por lo que es un buen momento de desplegar una primera versión en el entorno de pruebas. Tal y como se especificó en el apartado de Estado del arte, el entorno de pruebas

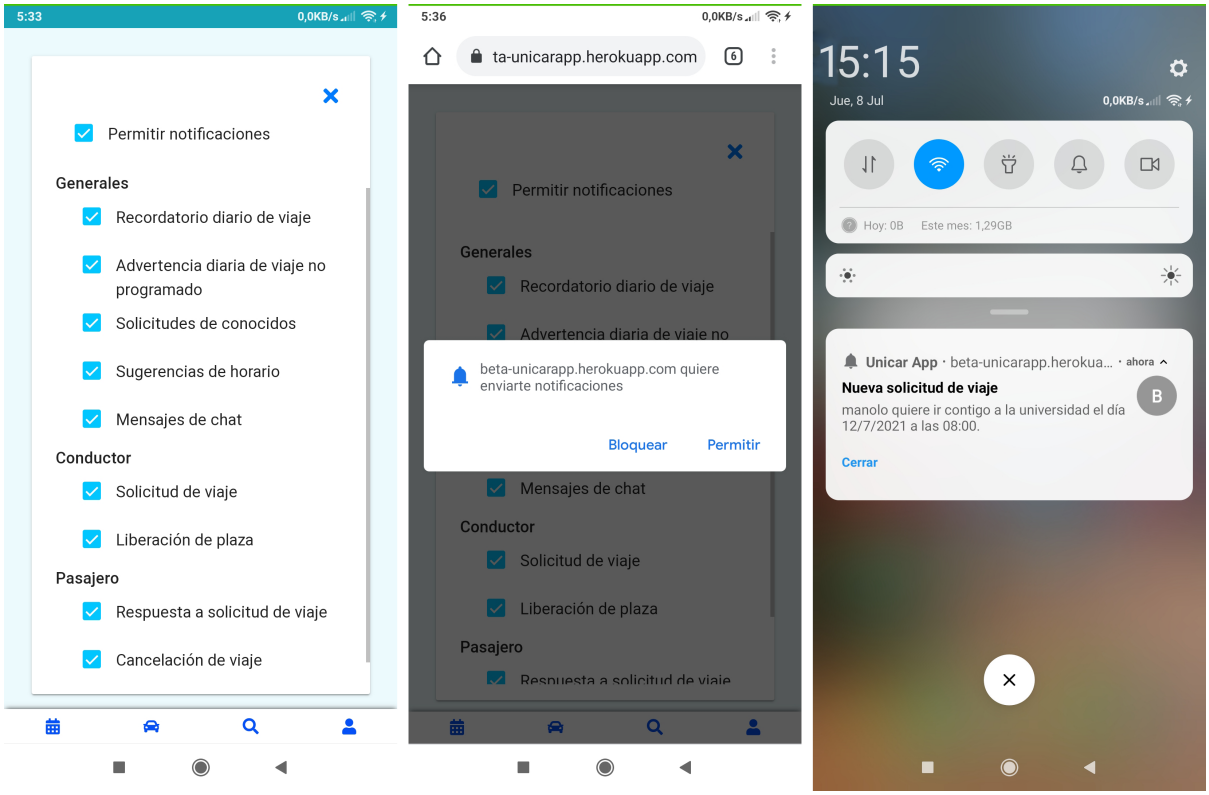


Figura 9.19: Capturas de pantalla del proceso de notificaciones
Fuente propia

será Heroku.

Lo primero a tener en cuenta para realizar el despliegue es que Heroku facilita este proceso haciendo uso del gestor de repositorios Git. Por esta razón, se crearán dos repositorios en GitHub, uno para la aplicación y otro para la API.

Una vez creados, desde la propia página web de Heroku, mediante una interfaz bastante intuitiva es posible desplegar aplicaciones. Tras crear una por cada proyecto, en la configuración de la aplicación, se debe asignar el repositorio que contenga el código del proyecto. Tras sincronizar el repositorio con la aplicación, al tratarse de proyectos de NodeJS, será suficiente con especificar el archivo de inicio y Heroku se encargará de ejecutarlo para compilar el proyecto y ponerlo en funcionamiento.

Ya con esta primera versión funcional de la aplicación desplegada en el entorno de prueba, podemos dar por finalizado el sprint y con ello, la etapa de implementación. Así pues los siguientes pasos consisten en probar y validar su buen funcionamiento y finalmente evaluar los resultados.

La sensación final tras esta etapa de implementación es buena, a pesar de no obtener la aplicación completa, como era previsible por su gran volumen. Más allá de los errores que pueden surgir durante las pruebas, la aplicación está prácticamente completa, a excepción de dos funcionalidades que se realizarán próximamente.

Una de ellas es la importación del calendario, cuya interfaz no está completa estéticamente, pero ha sido implementada la subida del archivo del calendario y el volcado de los datos del archivo en la base de datos. Sin embargo, las sugerencias de viaje no han sido posibles implementarlas por falta de tiempo. Se trata de una funcionalidad que implica mucho tiempo para pensar cómo implementarla y a ello se le suma el tiempo que conlleva desarrollarla. Estas dos, por tanto, serán dos futuras mejoras a realizar.

10. Pruebas y validación

En este apartado se detallarán las pruebas realizadas para validar el buen funcionamiento de la primera versión de la aplicación. Tal y como se ha especificado en el Diseño de pruebas y validación, esta versión ha sido compartida con un grupo reducido de personas, indicando los pasos para probar el funcionamiento de las cuatro funcionalidades principales. A continuación se muestra la lista de indicaciones:

- El primer paso consiste en registrar una cuenta en la aplicación. Para ello será necesario completar el formulario de registro y verificar la cuenta mediante el email enviado. Una vez terminado el proceso de registro, inicia sesión y accede a la página de perfil para rellenar los datos de usuario.

- El segundo paso consiste en publicar un viaje. Para ello, accede a la página de nuevo viaje (desde el menú principal o desde la sección de 'Mis Viajes'), rellena el formulario con los datos y pulsa el botón de 'Publicar Viaje'.

- El tercer paso consiste en buscar un viaje. Para ello, accede a la página de búsqueda desde el menú principal y rellena el formulario de búsqueda. Para poder probar totalmente esta funcionalidad se han introducido varios viajes de prueba con los siguientes datos: Jueves 08/07/2021 - IDA de Castalla a Universidad de Alicante, entre 8:00 y 10:00. y Jueves 08/07/2021 - VUELTA de Universidad de Alicante a Castalla, entre 13:00 y 15:00.

- El cuarto y último paso es agregar usuarios a la lista de conocidos. Para ello accede a la lista de conocidos desde la página de perfil, realiza una búsqueda y envía una solicitud. Como ejemplos de usuario, estos pueden ser: hectorest1, manolo, manolin y daniel25.

Además, junto a las instrucciones se ha incluido un enlace a un formulario de Google Forms en el que estos usuarios han tenido la posibilidad de evaluar la aplicación según su experiencia. Por cada paso de la lista anterior, se incluyen tres preguntas, dos de ellas obligatorias y otra opcional.

La primera preguntaba sobre si se ha podido realizar el proceso correctamente. La segunda, una valoración del 1 al 10 sobre la experiencia realizando el proceso. Y finalmente una tercera pregunta de libre contestación para aportar observaciones sobre la aplicación, especialmente en caso de no haber tenido una buena experiencia.

Concretamente se le ha ofrecido la posibilidad de probar la aplicación a un total de 15 personas, todas ellas de entre 18 y 25 años, de las cuales han participado 13, contabilizadas por el número de usuarios registrados en la base de datos. Sin embargo, teniendo en cuenta el número de respuestas al formulario para evaluarla, solamente han sido 8 las que lo han contestado.

En cuanto a los resultados de dicho formulario, estos han sido realmente positivos, pudiendo la gran mayoría de usuarios realizar todos los pasos correctamente. A continuación, se detallarán particularmente los resultados a las preguntas sobre cada funcionalidad.

Por lo que respecta al registro e inicio de sesión, el 87,5% de los usuarios, es decir, todos a excepción de 1, han podido registrarse en la aplicación, validar la cuenta y finalmente iniciar sesión. La valoración media de la experiencia obtenida en este paso ha sido de 9,87. Como observación, uno de los usuarios ha tenido un problema durante la verificación del usuario tras abrir la aplicación con otro navegador.

En el proceso de publicación de viaje, a pesar de poder todos los usuarios publicarlo correctamente, la valoración media obtenida ha sido inferior a la anterior. Esta ha sido de 9,25 y como observaciones, un usuario ha tenido problemas con la selección de la hora y en el momento de publicarlo no recibía ninguna respuesta, publicándolo varias veces.

La tercera funcionalidad ha resultado similar a la primera, con una persona que no ha conseguido buscar y solicitar un viaje correctamente. La valoración media en este caso ha sido 9,25. Como sugerencia en el proceso de búsquedas, varios usuarios han coincidido en que sería interesante poder buscar viajes sin especificar una hora o especificando un rango.

Finalmente, la última prueba ha resultado completamente exitosa, pudiendo todos los usuarios agregar correctamente a un conocido. La valoración ha sido de 9,25 en este caso y no ha habido ninguna sugerencia ni incidencia reseñable.

Como conclusión sobre los resultados obtenidos tras las pruebas, estos han sido muy favorables. Un aspecto trascendente es que los usuarios a los que se les ha proporcionado el acceso a la aplicación conocen la problemática que trata la aplicación. Su valoración positiva sobre la idea y funcionamiento de la aplicación es un indicativo de que el trabajo realizado no ha sido en vano.

11. Resultados

11.1. Producto final

Analizar el producto obtenido será determinante para valorar el éxito o fracaso del trabajo realizado. Sin embargo, por tratarse de un gran proyecto, este no ha sido implementado en toda su totalidad. Diferenciando el trabajo realizado entre la parte de cliente y servidor, a continuación se detallarán qué aspectos faltan por finalizar en cada una de ellas.

Sobre la parte del cliente, en la figura 11.1 se especificará la lista de interfaces que han sido implementadas y se indicará de cada una de ellas su estado actual. Los diferentes estados de las interfaces son: incompleta (rojo), revisión de errores (amarillo) y completa (verde). Además, algunas de ellas, por haber sido probadas en el proceso de Pruebas y validación están señaladas.

Como se puede observar, la gran mayoría de las interfaces han sido implementadas por completo. Y entre estas, el número de interfaces probadas por usuarios también es alto.

Sin embargo, a esta lista de interfaces, definida en el apartado de Diseño Interfaces se le debe añadir una con estado incompleto, correspondiente a la página con el enlace y la explicación sobre la instalación de la aplicación en un dispositivo.

Con respecto a la parte de servidor, la API ha sido completamente desarrollada. No obstante, hay tareas no finalizadas por falta de tiempo. Un ejemplo de ellas es la base de datos, que no ha sido completamente optimizada y que debería ser estudiada para introducir consultas almacenadas e índices para mejorar su rendimiento. Otro aspecto no tratado durante la implementación son las sugerencias de horario, cuya idea deberá analizarse y planificarse en un futuro.

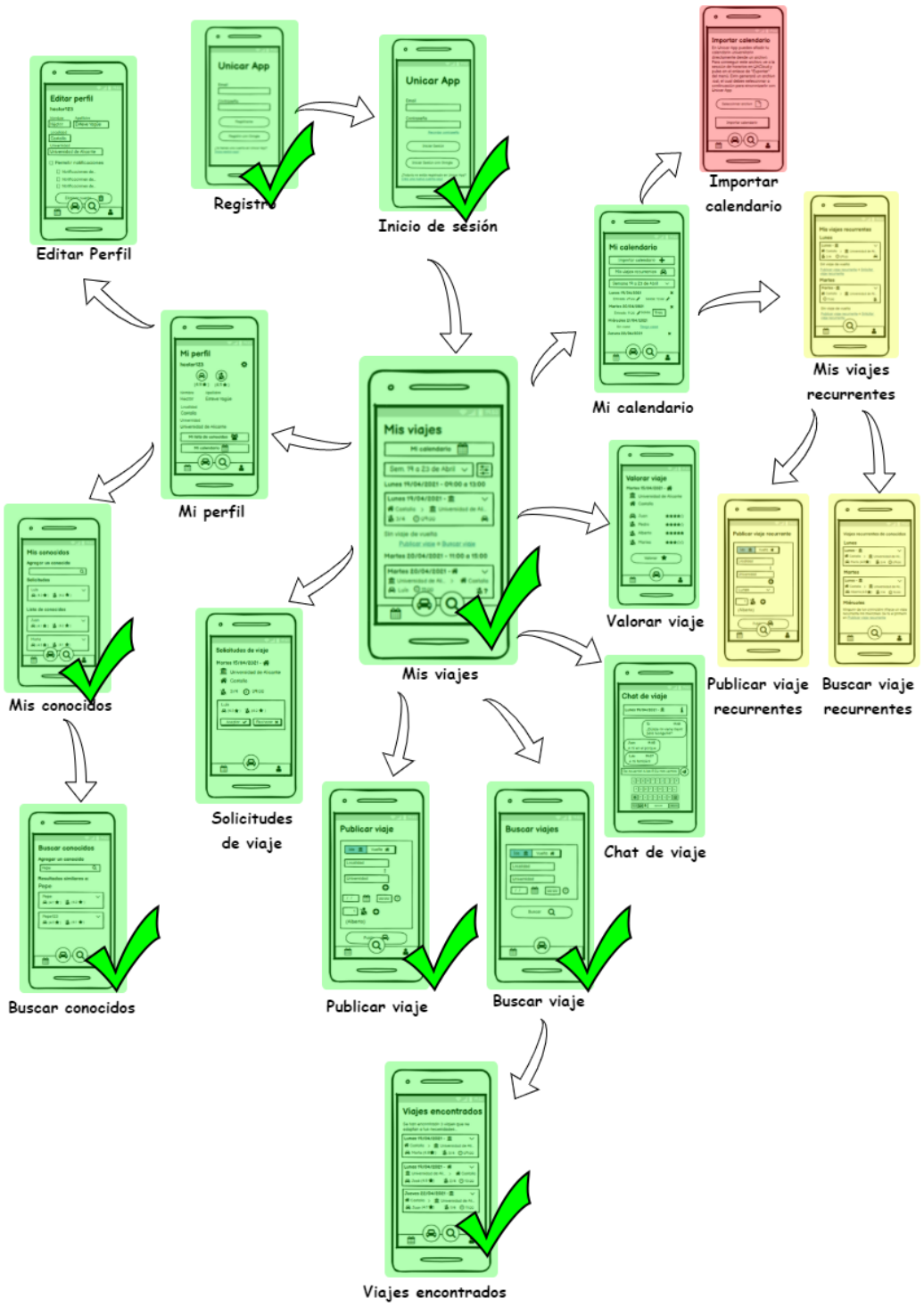


Figura 11.1: Interfaces completadas
Fuente propia

11.2. Costes temporales

Los costes temporales son otro de los aspectos a tener en cuenta una vez finalizado el proyecto para tener una referencia sobre el trabajo realizado y las tareas con más dedicación. Para tener controlado el tiempo dedicado al Trabajo de Fin de Grado, se ha utilizado la herramienta Toggl. Esta nos permite obtener resúmenes estadísticos sobre las horas trabajadas.

La primera estadística reseñable es el total de horas empleadas, y más concretamente distribuidas semanalmente. En la figura 11.2, se puede observar dicho reparto. En ella se aprecian a simple vista dos bloques.

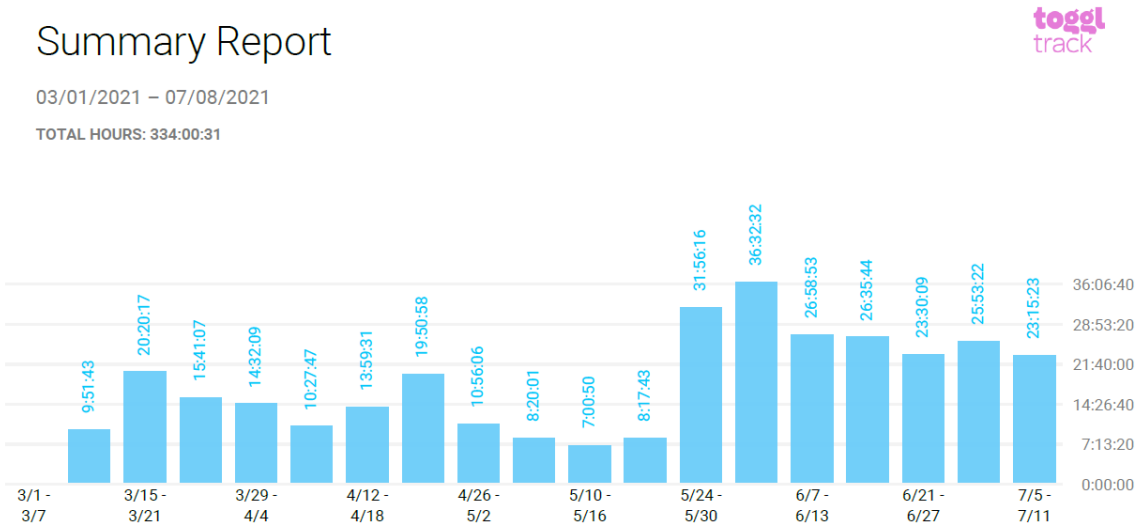


Figura 11.2: Resumen horas empleadas distribuidas por semana
Fuente: `https://toggl.com/`

El primer bloque, compuesto por las primeras once semanas, corresponde al desarrollo de los primeros apartados de la memoria, hasta la implementación, y, además, con el periodo en que estaba realizando las prácticas externas. El segundo, en cambio, está compuesto por las 7 últimas semanas y corresponde mayoritariamente a las horas dedicadas al desarrollo de la aplicación.

Entre ambos bloques se aprecia una diferencia notable entre las horas dedicadas semanalmente, pues en el segundo estas horas han aumentado considerablemente. Esto se debe a una mayor disponibilidad de tiempo, ya habiendo terminado el periodo de prácticas, y a la aproximación de la fecha de entrega.

Con respecto a esta fecha de entrega, cabe destacar que a falta de un mes para finalizar el plazo de entrega de la convocatoria C3 de Junio, se decidió cambiar la planificación y retrasar

la entrega para la convocatoria C4. De esta manera se dispondría de más tiempo para realizar un trabajo más completo y, sobretodo, sin realizar un exceso de horas en dicho mes.

La otra estadística destacable es las horas dedicadas a cada apartado de la memoria. Desde el primer momento, las tareas han sido registradas en esta herramientas diferenciándolas según el apartado de la memoria al que correspondían. Así pues, para realizar esta distinción se ha asignado un proyecto de Toggl para cada uno de los apartados y, además, uno relativo al desarrollo de la aplicación. El resultado se puede observar en la figura 11.3

Figura 11.3: Gráfica de horas por apartado
Fuente: `https://toggl.com/`

En esta gráfica se distinguen también dos bloques. El primero de ellos, en verde oscuro, se trata del desarrollo de la aplicación, cuyas horas dedicadas están incluidas conjuntamente. El resto de tareas son las relativas a la memoria, separadas por colores, como se ha mencionado anteriormente. Como curiosidad, destaca que prácticamente las horas dedicadas a ambos bloques están equilibradas.

Concretando en las tareas de la memoria, entre ellas sobresalen dos grupos. Estas son los relativos al Diseño y al Estado del arte, dos aspectos fundamentales previos a la implementación y que requieren una dedicación considerable para plantar las bases del proyecto.

En definitiva, el total de horas contabilizadas (334) se ajustan aproximadamente a las 300 horas que corresponden a una asignatura de 12 créditos, como es el TFG. Por esta razón, se puede concluir, que en este aspecto, el trabajo realizado ha sido el adecuado.

11.3. Asignaturas relacionadas

Finalmente, un tema que no podía pasar desapercibido en este Trabajo de Fin de Grado es el relativo a las asignaturas de la titulación relacionadas con el proyecto desarrollado. La titulación es Ingeniería Multimedia y dichas asignaturas son las siguientes, ordenadas según las etapas en que se han aplicados los conocimientos.

En cuanto al análisis de la viabilidad, planificación y metodologías, las principales asignaturas relacionadas son Proyectos Multimedia (cuarto curso) y Análisis y Especificación de Sistemas Multimedia (segundo curso).

Por lo que respecta al desarrollo de la aplicación, han resultado especialmente útiles los conocimientos y experiencia con los proyectos realizados en las asignaturas de Programación del Cliente Web (segundo curso) y Desarrollo de Aplicaciones Web (tercer curso).

Y finalmente, de cara a mejorar la experiencia del usuario, destacan los conocimientos obtenidos en Usabilidad y Accesibilidad (tercer curso) y Servicios Multimedia Avanzados (cuarto curso).

12. Conclusiones

Como último apartado del TFG, se expondrán a continuación las conclusiones finales y el trabajo relativo al proyecto a realizar en el futuro.

12.1. Evaluación de objetivos

Las primeras conclusiones a realizar están estrechamente relacionadas con los objetivos propuestos en el apartado de Objetivos. Si bien en su momento diferenciamos dos grupos de objetivos, estas conclusiones se realizarán sobre los relativos a los aspectos técnicos y desarrollo. Sobre el segundo grupo, por ser objetivos establecidos a largo plazo y ya con la aplicación completa, resulta imposible sacar conclusiones en este momento.

Los principales objetivos propuestos tenían por una parte la intención de implementar una aplicación web que pudiera imitar el comportamiento de una aplicación nativa, tanto por poder ser instalada en un dispositivo como por ofrecer la posibilidad de recibir notificaciones personalizadas.

En el momento en que fueron establecidos estos objetivos, el conocimiento sobre estos aspectos era notablemente inferior al actual, por lo que la investigación y el trabajo realizado ha sido productivo en ese aspecto. Por esta razón, lo que era un objetivo, ahora es una realidad, y tras este desarrollo se puede confirmar que una PWA puede simular a efectos prácticos de cara al usuario el comportamiento de una aplicación nativa.

Otro de los objetivos a cumplir era un bajo acoplamiento entre las tecnologías empleadas. Objetivo que, aunque todavía hay margen de mejora, podemos determinar como cumplido, ya que los dos grandes bloques del proyecto (Aplicación web y API+Base de datos) son totalmente independientes, comunicándose entre sí únicamente para realizar las peticiones relativas al acceso de datos.

Y como último objetivo, se trataba de dejar preparada la aplicación para que esta pudiera ser utilizada más allá de los componentes del grupo de WhatsApp. Este objetivo también se puede dar por hecho ya que es posible incluir diferentes localidades y diferentes Universidades.

12.2. Conclusiones personales

Una vez evaluados los objetivos y, dejando de lado si se han conseguido lograr, cabe realizar unas conclusiones más personales sobre esta etapa de desarrollo del proyecto.

La realidad es que, a pesar de coincidir con una mala época personalmente, realizar este trabajo me ha llevado a reforzar mi entusiasmo por este tipo de proyectos, resultando el tiempo invertido en el proyecto totalmente llevadero. Se trataba de una idea personal ya contemplada desde hace varios años, por lo que el concepto genérico estaba claro y esto ha sido fundamental de cara a avanzar en todo momento con pasos firmes.

Echando la vista atrás, y valorando todo el proceso, la conclusión sobre la experiencia es que, aunque los primeros pasos resultaban más pesados por ser únicamente desarrollo de la memoria, seguir este orden ha sido fundamental para haber llegado al momento de la implementación con la idea muy desarrollada y, sobretodo, sin haber implementado nada que después no resultara válido.

Además, han sido muchos los conocimientos adquiridos durante la elaboración de este Trabajo de Final de Grado, por lo que las horas invertidas han sido realmente productivas.

Sin duda, considero este trabajo realizado como un buen culmen finalizar mi etapa como estudiante de esta titulación. Un proyecto que me ha aportado mucha confianza de cara a ser consciente de mis capacidades.

12.3. Líneas de trabajo futuro

Como líneas de trabajo futuro, en primera instancia la idea es continuar con el desarrollo del proyecto para terminar con las funcionalidades que han faltado implementar. Puesto que todavía hay objetivos por cumplir a largo plazo, y estos están relacionados con el despliegue en el entorno de producción, será necesario continuar con la fase de pruebas y validación para pulir todos los detalles.

Sin embargo, han surgido dos ideas que le pueden aportar valor, más allá de ajustar las funcionalidades contempladas en el proyecto. Estos se recogen a continuación:

Por una parte, complementar el uso de la aplicación con redes sociales. Por ejemplo, poder publicar en redes sociales un mensaje en el que se comparta la información de un viaje publicado. De esta manera, indirectamente se daría visibilidad a la aplicación, pudiendo así captar más usuarios.

Por otra parte, añadir la posibilidad de crear grupos. Además de la opción de añadir conocidos particularmente, crear grupos de conocidos en los que los viajes serían privados

para cada grupo.

En definitiva, la intención es continuar aportando el mayor valor posible a la aplicación para que el trabajo realizado durante el Trabajo de Final de Grado sea tan solo el inicio de un largo camino.

Además, en vistas a los objetivos en producción, paralelamente al desarrollo, se empezará a trabajar la difusión de la marca a través de redes sociales y la web corporativa. De este modo, cuando la aplicación esté lista, los usuarios potenciales ya serán conocedores de su existencia y les resultará más interesante probarla.

Bibliografía

4 Examples of UX Personas. (2021, Apr). Descargado de `https://qubstudio.com/blog/4-examples-of-ux-personas` ([Online; accessed 24. May 2021])

ACID - PiperLab. (2020, Oct). Descargado de `https://piperlab.es/glosario-de-big-data/acid` ([Online; accessed 7. May 2021])

Axarnet. (2021, Apr). *¿Qué son IaaS, PaaS, SaaS? Conoce las diferencias.* Descargado de `https://axarnet.es/blog/saas-paas-iaas` ([Online; accessed 7. Apr. 2021])

Bases de datos relacionales vs. no relacionales: ¿qué es mejor? - Aukera. (2018, Sep). Descargado de `https://aukera.es/blog/bases-de-datos-relacionales-vs-no-relacionales` ([Online; accessed 7. May 2021])

Carmona, J. A. (2019, Dec). ¿Son el futuro las Aplicaciones Web Progresivas? ¿Enterrarán definitivamente a las aplicaciones nativas? *Xataka Windows.* Descargado de `https://www.xatakawindows.com/actualidad-en-redmond/son-el-futuro-las-aplicaciones-web-progresivas-enterraran-definitivamente-a-las-aplicaciones-nativas` ([Online; acceso 24. Mar. 2021])

ClearDB MySQL | Heroku Dev Center. (2021, Apr). Descargado de `https://devcenter.heroku.com/articles/cleardb` ([Online; accessed 6. Apr. 2021])

Code, F. (2020, Jan). *Web Push | Notificaciones usando Nodejs y Service Workers.* Youtube. Descargado de `https://www.youtube.com/watch?v=B3O__1IiVIY` ([Online; accessed 30. Mar. 2021])

Coolors - The super fast color schemes generator! (2021, Jun). Descargado de `https://coolors.co` ([Online; accessed 17. Mar. 2021])

Customer Journey Map o Mapa de Experiencia del Cliente + Ejemplo y Vídeo. (2021, Apr). Descargado de `https://innokabi.com/claves-para-emocionar-a-tu-cliente-customer-journey-map` ([Online; accessed 24. May 2021])

Especificación de Requisitos según el estándar de IEEE 830. (2018, Oct). Descargado de `https://moodle2019-20.ua.es/pensemonline/pluginfile.php/10183/mod_resource/content/2/IEEE830_esp.pdf` ([Online; accessed 17. Apr. 2021])

Express - Infraestructura de aplicaciones web Node.js. (2021, Apr). Descargado de `https://expressjs.com/es` ([Online; accessed 6. Apr. 2021])

Express Web Framework (Node.js/JavaScript) - Aprende sobre desarrollo web | MDN. (2021, Apr). Descargado de https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs ([Online; accessed 6. Apr. 2021])

Fazt. (2017, Aug). *¿Que es Node.js? ¿Para que sirve?* Youtube. Descargado de https://www.youtube.com/watch?v=9U8EaVjuq6U ([Online; accessed 1. Apr. 2021])

Fazt. (2018, Apr). *Chat Javascript Desde Cero | Crea un Chat con HTML5, Nodejs, Express, WebSockets y Mongodb, Parte 1.* Youtube. Descargado de https://www.youtube.com/watch?v=vGikkrp-HPM ([Online; accessed 30. Mar. 2021])

Gaunt, M. (2020, Jul). Introducción a los service workers | Web Fundamentals. *Google Developers.* Descargado de https://developers.google.com/web/fundamentals/primers/service-workers?hl=es ([Online; acceso 30. Mar. 2021])

The Good and the Bad of Node.js Web App Development. (2021, Mar). Descargado de https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development ([Online; accessed 1. Apr. 2021])

HackaBoss. (2020, May). 5 ejemplos de aplicaciones Node.js para empresas. *HACK A BOSS.* Descargado de https://hackaboss.com/blog/5-mejores-ejemplos-de-aplicaciones-node-js-para-empresas ([Online; accessed 6. Apr. 2021])

Introduction to progressive web apps - Progressive web apps (PWAs) | MDN. (2021, Mar). Descargado de https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#advantages_of_web_applications ([Online; acceso 30. Mar. 2021])

Is Node.js really better than PHP? Let's go Through the Pros and Cons. (2021, Apr). Descargado de https://datafloq.com/read/is-nodejs-better-than-php-pros-cons/5297 ([Online; accessed 1. Apr. 2021])

Kadam, S. (2019, Jan). Pros and Cons Of Node.Js For Web Development - Swati Kadam - Medium. *Medium.* Descargado de https://medium.com/@swan2903/pros-and-cons-of-node-js-for-web-development-831b3b4baa04

Knöbl, E. (2021, Apr). *Objetivos SMART: qué son y cómo utilizarlos.* Descargado de https://www.titular.com/blog/objetivos-smart-que-son-y-como-utilizarlos ([Online; accessed 15. Apr. 2021])

Konathala, S. (2020, Jun). A World Without Node.js? - TechInPieces - Medium. *Medium.* Descargado de https://medium.com/techinpieces/a-world-without-node-js-12fec4b18733

Las 5 ceremonias Scrum: claves para la gestión de procesos. (2021, Apr). Descargado de https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html ([Online; accessed 15. Apr. 2021])

López, S. (2020, Jun). Qué es PWA: características, ventajas y desventajas. *DIGITAL55*. Descargado de https://www.digital55.com/desarrollo-tecnologia/que-es-pwa-ventajas-desventajas ([Online; acceso 30. Mar. 2021])

Manejo de errores de Express. (2021, Apr). Descargado de https://expressjs.com/es/guide/error-handling.html ([Online; accessed 6. Apr. 2021])

Node JS Ventajas y Desventajas. Desarrollo | Openinnova. (2019, Jul). *Openinnova*. Descargado de https://www.openinnova.es/node-js-ventajas-y-desventajas-desarrollo ([Online; accessed 1. Apr. 2021])

Objetivos SMART: Define tus metas sin cometer errores. (2019, May). Descargado de https://blogsterapp.com/es/objetivos-smart-definir-metas ([Online; accessed 8. Apr. 2021])

OpenWebinars. (2018, Feb). *QUÉ ES EXPRESS.* Youtube. Descargado de https://www.youtube.com/watch?v=OMzOK7V0k3Q ([Online; accessed 1. Apr. 2021])

Osmani, A. (2019, May). The App Shell Model | Web Fundamentals | Google Developers. *Google Developers*. Descargado de https://developers.google.com/web/fundamentals/architecture/app-shell ([Online; acceso 30. Mar. 2021])

Procedimientos MySQL: ventajas, desventajas y casos de uso - ADN Cloud. (2019, May). Descargado de https://blog.mdcloud.es/procedimientos-mysql-ventajas-desventajas ([Online; accessed 23. Jun. 2021])

Programación por capas. (2014, May). Descargado de https://www.virtuniversidad.com/greenstone/collect/informatica/archives/HASH0195.dir/doc.pdf ([Online; accessed 10. May 2021])

Ramírez, I. (2018, Jul). ¿Qué es una Aplicación Web Progresiva o PWA? *Xataka*. Descargado de https://www.xataka.com/basics/que-es-una-aplicacion-web-progresiva-o-pwa ([Online; acceso 24. Mar. 2021])

Redacción APD. (2020, May). ¿En qué consiste la metodología Kanban y cómo utilizarla? *APD España*. Descargado de https://www.apd.es/metodologia-kanban ([Online; accessed 14. Apr. 2021])

RyteWiki. (2021, Jan). *¿Qué es el Mobile First? - Ryte Digital Marketing Wiki.* Descargado de https://es.ryte.com/wiki/Mobile_First ([Online; accessed 14. Apr. 2021])

Sánchez, James. (2014, Sep). *Metodologías ágiles de desarrollo en el trabajo individual.* Descargado de https://www.freelancer.es/community/articles/metodologias-agiles-desarrollo-trabajo-individual ([Online; accessed 14. Apr. 2021])

¿Cuál es la metodología más adecuada para tu proyecto? (2021, Apr). Descargado de https://www2.deloitte.com/es/es/pages/technology/articles/waterfall-vs-agile.html ([Online; accessed 14. Apr. 2021])

Web Survey Report 2018 | Node.js. (2020, Feb). Descargado de `https://nodejs.org/en/user-survey-report/#Types-of-Development-Work` ([Online; accessed 31. Mar. 2021])

What are some alternatives to Node.js? - StackShare. (2021, Mar). Descargado de `https://stackshare.io/nodejs/alternatives` ([Online; accessed 31. Mar. 2021])

What is Docker? (2021, Apr). Descargado de `https://opensource.com/resources/what-docker` ([Online; accessed 7. Apr. 2021])

A. Fragmentos de código

Código A.1: Archivo index.js

```
const express = require('express');
const app = express();
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, ↩
      ↪ Authorization");
  res.header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE");
  next();
});

// Settings
app.set('port', process.env.PORT || 3050);

// Middlewares
app.use(express.json());

// Routes
const router = require('./routes')

app.use('/api', router);

// Starting the server
app.listen(app.get('port'), () => {
  console.log(`Server on port ${app.get('port')}`);
});
```

Código A.2: Archivo database.js

```
const mysql = require('mysql');
const mysqlConnection = mysql.createConnection({
  host: '********',
  user: '********',
  password: '********',
  database: '********',
  multipleStatements: true
});

mysqlConnection.connect(function (err) {
  if (err) {
    console.error(err);
    return;
  } else {
    console.log('db is connected');
  }
});
```

```
18
19  module.exports = mysqlConnection;
20  });
```

Código A.3: Estructura archivo routes/index.js

```
1   const express = require('express');
2   const router = express.Router();
3
4   const usuariosCtrl = require('../controllers/usuarios')
5   const viajes_recurrentesCtrl = require('../controllers/viajes_recurrentes')
6   const viajesCtrl = require('../controllers/viajes')
7   const ubicacionesCtrl = require('../controllers/ubicaciones')
8   const adminCtrl = require('../controllers/admin')
9
10  // RUTAS USUARIOS
11  router.get('/usuarios', usuariosCtrl.getUsuarios)
12  router.get('/usuarios/:id', usuariosCtrl.getUsuario)
13  .
14  .
15  .
16  router.delete('/viajes/:id/solicitudes/:id_usuario', viajesCtrl.deleteSolicitudViaje)
17
18  module.exports = router;
```

Código A.4: Ejemplo estructura controller (controllers/viajes.js)

```
1   const mysqlConnection = require('../database.js');
2   const util = require('util');
3
4   const mysqlquery = util.promisify(mysqlConnection.query).bind(mysqlConnection);
5
6   // GET /viajes : Devuelve los viajes que coinciden con los filtros
7   // GET /viajes?idUsuario={id_usuario} : Devuelve la lista de viajes del usuario
8   async function getViajes(req, res) {
9
10  }
11
12  // GET /viajes/:id : Devuelve un viaje por id
13  async function getViaje (req, res){
14
15  }
16  .
17  .
18  .
19  // DELETE /viajes/:id/solicitudes/:id_usuario : Elimina el registro de la plaza del usuario en↩
        ↪  el viaje a partir de sus respectivos id.
20  async function deleteSolicitudViaje(req, res){
21
22  }
23
24  module.exports = {
25    getViajes,
26    getViaje,
27    getSolicitudesViaje,
28    getUsuariosViaje,
29    getChatViaje,
30    postViaje,
```

```
    postSolicitudViaje,
    postMensajeViaje,
    postValoracionViaje,
    updateViaje,
    contestarSolicitudViaje,
    deleteViaje,
    deleteSolicitudViaje
};
```

Código A.5: Ejemplo consulta a base de datos

```
// GET /viajes/:id : Devuelve un viaje por id
async function getViaje (req, res){
  const { id } = req.params;
  var query = "SELECT ...
              FROM ...
              WHERE ...";

  try {
    rows = await mysqlquery(query)

    res.status(200).json({
      "viaje": rows[0]
    });
  } catch(err){
    res.status(500).json({
        "error": err
    });
  }
};
```

Código A.6: Primera versión archivo index.html

```
<!DOCTYPE html>
<html lang="es" ng-app="app">
<head>
    <!-- Cabecera -->
</head>

<body layout="column">
    <div ng-view id="main"></div>

    <!-- Importación de librerías de terceros -->
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

    <!-- Servicios para consultas a API -->
    <script src="./services/viajes.js"></script>

    <!-- Importación de controladores -->
    <script src="./controllers/misviajesController.js"></script>

    <!-- Módulo de la aplicación con rutas -->
    <script src="app.js"></script>
</body>
</html>
```

Código A.7: Primera versión archivo app.js

```
var app = angular.module("app", [
    'app.factories.viajes',
    'app.misviajes',
    'ngRoute',
    ]).config(config);

config.$inject = ['$routeProvider', '$locationProvider']

// RUTAS

function config($routeProvider, $locationProvider){
    $routeProvider
    .when("/misviajes", {
        templateUrl: "./views/misviajes.html",
        controller: "misviajesController as mvsCtrl"
    })
    .otherwise({
        redirectTo: '/'
    });

    $locationProvider.html5Mode(true);
}
```

Código A.8: Estructura archivo "services/viajes.js"

```
    angular
        .module('app.factories.viajes', [])
        .factory('Viajes', Viajes);

    Viajes.$inject = ['$http', '$cookies', '$window'];

    function Viajes($http, $cookies, $window){
        var self = this;

        var unicarapp_token = $window.sessionStorage.unicarapp_token;
        $http.defaults.headers.common.Authorization = `bearer ${unicarapp_token}`

        self.getViaje = getViaje;

        function getViaje(id) {

            var url = `http://localhost:3050/api/viajes/${id}`

            return $http.get(url)
                .then(res=>{
                    return res.data
                })
        }

        return self
    }
```

Código A.9: Generación y comprobación de contraseña con bcrypt

```
//Encriptar contraseña
  function generateHash(password) {
    return new Promise(res => {
      bcrypt.genSalt(10, (err, salt)=>{
        bcrypt.hash(password, salt, null, (err, hash)=>{
          res(hash);
        })
      })
    });
  }

//Comprobación de contraseña
  const password_verification = bcrypt.compareSync(
    req.body.password,
    usuario.password
  );
```

Código A.10: Autorización basada en token

```
//Creación de token en "services/index.js"
  function createToken (user) {

      if(!user.admin){
          user.admin = false;
      }

    const payload = {
      sub: user.id,
      iat: moment().unix(),
      exp: moment().add(14, 'days').unix(),
      admin: user.admin
    }

    return jwt.encode(payload, config.SECRET_TOKEN)
  }

//Decodificación del token en "services/index.js"
 function decodeToken (token) {
    const decoded = new Promise((resolve, reject) => {
      try{
       const payload = jwt.decode(token, config.SECRET_TOKEN)

       if(payload.exp <= moment().unix()){
         reject({
          status: 401,
          message: 'El token ha expirado'
         })
       }
       resolve(payload)
      }catch(err){
        reject({
          status: 500,
          message: 'Invalid token'
        })
      }
    })

    return decoded
  }

```

```
//Proceso de autorización de usuario en "middleware/auth.js"
    function isAuth(req, res, next) {
     if (!req.headers.authorization) {
       return res.status(403).send({message: 'No tienes autorización'});
     }

     const token = req.headers.authorization.split(' ')[1];

     services
       .decodeToken(token)
       .then((response) => {
         req.user = response;
         next();
       })
       .catch((response) => {
         res.status(response.status);
       });
   }
```

Código A.11: Cabeceras para PWA

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">

 <meta name="theme-color" content="#FFFFFF">
 <meta name="MobileOptimized" content="width">
 <meta name="HandheldFriendly" content="true">
 <meta name="apple-mobile-web-app-captable" content="yes">
 <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
 <link rel="shortcut icon" type="image/png" href="./img/logo_16.png">
 <link rel="apple-touch-icon" href="./img/logo_16.png">
 <link rel="apple-touch-startup-image" href="./img/logo_16.png">
 <link rel="manifest" href="manifest.json">
```

B. Diseño de API

En este anexo se incluye el documento de especificación de la API. Este documento está compuesto por todas las rutas que atenderán las peticiones a la capa de datos hechas por la capa del cliente. Así pues, se especifica de cada ruta el verbo correspondiente al tipo de petición (GET, POST, PUT y DELETE), una descripción, sus parámetros, el cuerpo a enviar, en los casos de las peticiones POST y PUT, y la respuesta que la API devolverá en estado correcto.

Especificación de la API REST

Usuarios

GET	/usuarios

Descripción: Devuelve la lista de usuarios que coinciden con los parámetros
Parámetros: nombre_usuario

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "usuarios": [
       {
         "id": integer,
         "nombre_usuario": string,
         "foto_perfil": string,
         "val_usuario_conductor": float,
         "val_usuario_pasajero": float,
         "localidad": string,
         "universidad": string
       },
       ...
   ]
}
```

GET	/usuarios/{id}

Descripción: Devuelve los datos del usuario pasado como parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "usuario":
       {
         "id": integer,
         "email": string,
         "nombre_usuario": string,
         "nombre": string,
         "apellidos": string,
         "edad": string,
```

```
        "foto_perfil": string,
        "verificado": boolean,
        "val_usuario_conductor": float,
        "val_usuario_pasajero": float,
        "localidad": string,
        "universidad": string
      }
}
```

GET	/usuarios/{id}/conocidos

Descripción: Devuelve la lista de usuarios conocidos por el usuario pasado por parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "conocidos": [
      {
        "id": integer,
        "email": string,
        "nombre_usuario": string,
        "nombre": string,
        "apellidos": string,
        "foto_perfil": string,
        "val_usuario_conductor": integer,
        "val_usuario_pasajero": integer,
        "localidad": string,
        "universidad": string
      },
      ...
   ]
}
```

GET	/usuarios/{id}/conocidos/solicitudes

Descripción: Devuelve la lista de solicitudes de usuarios para ser conocidos del usuario pasado por parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "solicitudes": [
       {
         "id": integer,
         "email": string,
         "nombre_usuario": string,
         "nombre": string,
         "apellidos": string,
         "foto_perfil": string,
         "val_usuario_conductor": integer,
         "val_usuario_pasajero": integer,
         "localidad": string,
         "universidad": string
       },
       ...
   ]
}
```

GET	/usuarios/{id}/calendario

Descripción: Devuelve la lista de días del usuario pasado por parámetro y que, además, coinciden con los parámetros.
Parámetros: fecha_min, fecha_max

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "dias": [
       {
         "fecha": date,
         "hora_entrada": time,
         "hora_salida": time
       },
       ...
   ]
}
```

GET	/usuarios/{id}/suscripciones_notificaciones

Descripción: Devuelve la lista de notificaciones a las que está suscrito el usuario.

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "suscripciones": {
        "permNot": boolean,
        "recViaje": boolean,
        "solCon": boolean,
        "sugHor": boolean,
        "chat": boolean,
        "solVia": boolean,
        "libPla": boolean,
        "respSol": boolean
      }
}
```

POST /usuarios

Descripción: Crea un usuario en la base de datos del registro. Inicialmente el usuario no estará verificado.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "email": string,
 "nombre_usuario": string,
 "password": string
}
```

POST /usuarios/{id}/conocidos/{id_usuario}

Descripción: Registra una solicitud de conocido del usuario pasado en el primer parámetro al usuario pasado en el segundo parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

POST /usuarios/{id}/calendario

Descripción: Inserta en la base de datos un registro de un día del usuario pasado por parámetro

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "fecha": date,
 "hora_entrada": datetime,
 "hora_salida": datetime,
}
```

POST /usuarios/{id}/notificaciones

Descripción: Activa las notificaciones del usuario

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "suscripcion": object
}
```

POST /usuarios/{id}/notificaciones/{id_notificacion}

Descripción: Suscribe al usuario al tipo de notificación especificado por parámetro

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

PUT	/usuarios/{id}

Descripción: Modifica la información del usuario con el id pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "nombre": string,
 "apellidos": string,
 "edad": string,
 "foto_perfil": string
}
```

PUT	/usuarios/{id}/conocidos/{id_usuario}

Descripción: Cambia el estado de la solicitud del usuario pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "estado": integer
}
```

PUT	/usuarios/{id}/calendario/{fecha}

Descripción: Modifica los datos del día pasado como parámetro del usuario pasado por parámetro

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "hora_entrada": datetime,
 "hora_salida": datetime,
}
```

DELETE	/usuarios/{id}

Descripción: Elimina el usuario con el id pasado como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

DELETE	/usuarios/{id}/conocidos/{id_usuario}

Descripción: Elimina la relación de conocidos entre el usuario pasado por el primer parámetro y el usuario pasado por el segundo parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

DELETE	/usuarios/{id}/calendario/{fecha}

Descripción: Elimina el día pasado como parámetro del usuario pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

DELETE /usuarios/{id}/notificaciones/

Descripción: Desactiva las notificaciones del usuario.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

DELETE /usuarios/{id}/notificaciones/{id_notificacion}

Descripción: Borra la suscripción del usuario al tipo de notificaciones pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

Viajes

GET /viajes

Descripción: Devuelve la lista de viajes ofertados que coinciden con los parámetros
Parámetros: sentido, fecha, hora_salida, hora_llegada, origen, destino

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viajes": [
       {
         "id": integer,
         "sentido": integer,
```

```
        "fecha": date,
        "hora_salida": time,
        "hora_llegada": time,
        "origen": string,
        "destino": string,
        "id_usuario": integer,
        "nombre_usuario": string,
        "val_usuario_conductor": integer,
        "plazas": integer,
        "plazas_ocupadas": integer,
      },
      ...
  ]
}
```

GET	/viajes?idUsuario={idUsuario}

Descripción: Devuelve la lista de viajes del usuario pasado en la consulta y que, además, coinciden con los demás filtros
Parámetros: sentido, fecha_min, fecha_max, origen, destino, rol, estado

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viajes": [
      {
        "id": integer,
        "sentido": integer,
        "fecha": date,
        "hora_salida": time,
        "hora_llegada": time,
        "origen": string,
        "destino": string,
        "id_usuario": integer,
        "nombre_usuario": string,
        "plazas": integer,
        "plazas_ocupadas": integer,
        "estado": integer,
        "rol": integer,
      },
      ...
  ]
}
```

GET	/viajes/{id}

Descripción: Devuelve los datos del viaje con el id pasado como parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viaje":
       {
         "id": integer,
         "sentido": integer,
         "fecha_salida": datetime,
         "fecha_salida": datetime,
         "origen": string,
         "destino": string,
         "id_usuario": integer,
         "nombre_usuario": string,
         "plazas": integer,
         "plazas_ocupadas": integer,
         "val_usuario_conductor": integer
       }
}
```

GET	/viajes/{id}/solicitudes

Descripción: Devuelve la lista de solicitudes con estado pendiente del viaje con el id pasado como parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "solicitudes":
       [{
         "id": integer,
         "fecha": datetime,
         "id_usuario": integer,
         "nombre_usuario": string,
         "val_usuario_conductor": float
         "val_usuario_pasajero": float
       },
       ...
       ]
}
```

GET	/viajes/{id}/usuarios?idUsuario={idUsuario}

Descripción: Devuelve la lista de usuarios del viaje con el id pasado como parámetro a excepción del que realiza la petición pasado por parámetro en la query

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "usuarios": [
       {
         "id": integer,
         "id_usuario": integer,
         "nombre_usuario": string,
         "val_usuario_conductor": float
         "val_usuario_pasajero": float
       },
       ...
    ]
}
```

GET	/viajes/{id_viaje}/chat

Descripción: Devuelve la lista de mensajes del chat del viaje pasado como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "mensajes": [
       {
         "id": integer,
         "fecha": datetime,
         "contenido": string,
         "usuario": integer,
         "nombre_usuario": string,
       },
       ...
    ]
}
```

POST	/viajes

Descripción: Crea un viaje en la base de datos. El usuario conductor del viaje será el asociado al token con el que se autoriza la petición.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "sentido": integer,
 "fecha": datetime,
 "hora_salida": datetime,
 "hora_llegada": datetime,
 "origen": integer,
 "destino": integer,
 "plazas": integer,
 "recurrente": integer,
}
```

POST	/viajes/{id}/solicitudes

Descripción: Crea un registro de la solicitud del usuario que realiza la petición (según el token) al viaje pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "usuario": integer
}
```

POST	/viajes/{id_viaje}/chat

Descripción: Crea un mensaje en la base de datos asociado al viaje pasado como parámetro. El usuario emisor será el que asociado al token con el que se autoriza la petición.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "usuario": int,
 "contenido": string
}
```

POST /viajes/{id_viaje}/valoracion

Descripción: Inserta en la base de datos las valoraciones del usuario que realiza la petición (según el token) sobre los usuarios del viaje pasado como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
  "usuario": integer,
  "valoraciones" : [
    {
      "valorado": integer,
      "valoracion": integer,
      "comentario": string,
      "rol_valorado": integer
    },
    ...
  ]
}
```

PUT /viajes/{id}

Descripción: Modifica la información del viaje pasado como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "hora_salida": time,
 "hora_llegada": time,
 "origen": integer,
 "destino": integer,
 "plazas": integer
}
```

PUT /viajes/{id}/solicitudes/{id_usuario}

Descripción: Modifica el estado de la solicitud del usuario al viaje con las respectivas id pasadas como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "estado": integer
}
```

DELETE /viajes/{id}

Descripción: Elimina el viaje con el id pasado como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

DELETE	/viajes/{id}/solicitudes/{id_usuario}

Descripción: Elimina el registro de la plaza del usuario en el viaje a partir de sus respectivos id.

```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

Viajes recurrentes

GET	/viajes_recurrentes

Descripción: Devuelve la lista de viajes recurrentes de los usuarios conocidos por el usuario que realiza la petición (según el token) y que, además, coinciden con los parámetros.
Parámetros: sentido, dia_semana, origen, destino

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viajes_recurrentes": [
       {
         "id": integer,
         "sentido": integer,
         "dia_semana": integer,
         "hora_salida": string,
         "hora_llegada": string,
         "origen": string,
         "destino": string,
         "id_usuario": integer,
         "nombre_usuario": string,
         "val_usuario_conductor": string,
         "plazas": integer,
         "plazas_ocupadas": integer,
       },
       ...
   ]
}
```

GET	/viajes_recurrentes?idUsuario={idUsuario}

Descripción: Devuelve la lista de viajes recurrentes del usuario pasado en la consulta y que, además, coinciden con los parámetros.
Parámetros: sentido, dia_semana, origen, destino

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viajes_recurrentes": [
       {
         "id": integer,
         "sentido": integer,
         "dia_semana": integer,
         "hora_salida": datetime,
         "hora_llegada": datetime,
         "origen": string,
         "destino": string,
         "plazas": integer,
         "plazas_ocupadas": integer,
         "rol": integer
       },
       ...
   ]
}
```

GET	/viajes_recurrentes/{id}

Descripción: Devuelve los datos del viaje recurrente con el id pasado como parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "viaje_recurrente":
       {
         "id": integer,
         "sentido": integer,
         "fecha_salida": datetime,
         "fecha_llegada": datetime,
         "origen": string,
         "destino": string,
         "id_usuario": integer,
         "nombre_usuario": string,
         "plazas": integer,
```

```
        "plazas_ocupadas": integer,
      }
}
```

GET /viajes_recurrentes/{id}/solicitudes

Descripción: Devuelve la lista de solicitudes con estado pendiente del viaje recurrente con el id pasado como parámetro

```
{
  "descripcion": "Éxito",
  "estado": 200,
  "solicitudes":
      [{
        "id": integer,
        "fecha": datetime,
        "id_usuario": integer,
        "nombre_usuario": string,
        "val_usuario_conductor": float
        "val_usuario_pasajero": float
      },
      ...
      ]
}
```

POST /viajes_recurrentes

Descripción: Crea un viaje recurrente en la base de datos. El usuario conductor del viaje será el asociado al token con el que se autoriza la petición.

```
{
  "descripcion": "Éxito",
  "estado": 201,
}
```

Body:

```
{
 "sentido": integer,
 "fecha_salida": datetime,
 "fecha_salida": datetime,
 "origen": integer,
```

```
 "destino": integer,
 "id_usuario": integer,
 "plazas": integer,
 "viaje_recurrente": integer,
}
```

POST /viajes_recurrentes/{id}/solicitudes

Descripción: Crea un registro de la solicitud del usuario que realiza la petición (según el token) al viaje recurrente pasado por parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "usuario": integer
}
```

PUT /viajes_recurrentes/{id}/solicitudes/{id_usuario}

Descripción: Modifica el estado de la solicitud del usuario pasado por parámetro al viaje recurrente con las respectivas id pasadas como parámetro.

```
{
  "descripcion": "Éxito",
  "estado": 201
}
```

Body:

```
{
 "estado": integer
}
```

DELETE /viajes_recurrentes/{id}

Descripción: Elimina el viaje recurrente con el id pasado como parámetro.


```
{
  "descripcion": "Éxito",
  "estado": 200
}
```


DELETE	/viajes_recurrentes/{id}

Descripción: Elimina el registro de la plaza del usuario que realiza la petición (según el token) en el viaje con la id .


```
{
  "descripcion": "Éxito",
  "estado": 200
}
```

C. Diseño de Interfaces

El siguiente anexo es la lista de wireframes diseñados como guía para la implementación de las diferentes interfaces de la aplicación. Por cada una de ellas se detallan los requisitos funcionales que cubren y una breve descripción de su funcionamiento y/o comportamiento.

Diseño de Interfaces

Wireframes con Balsamiq

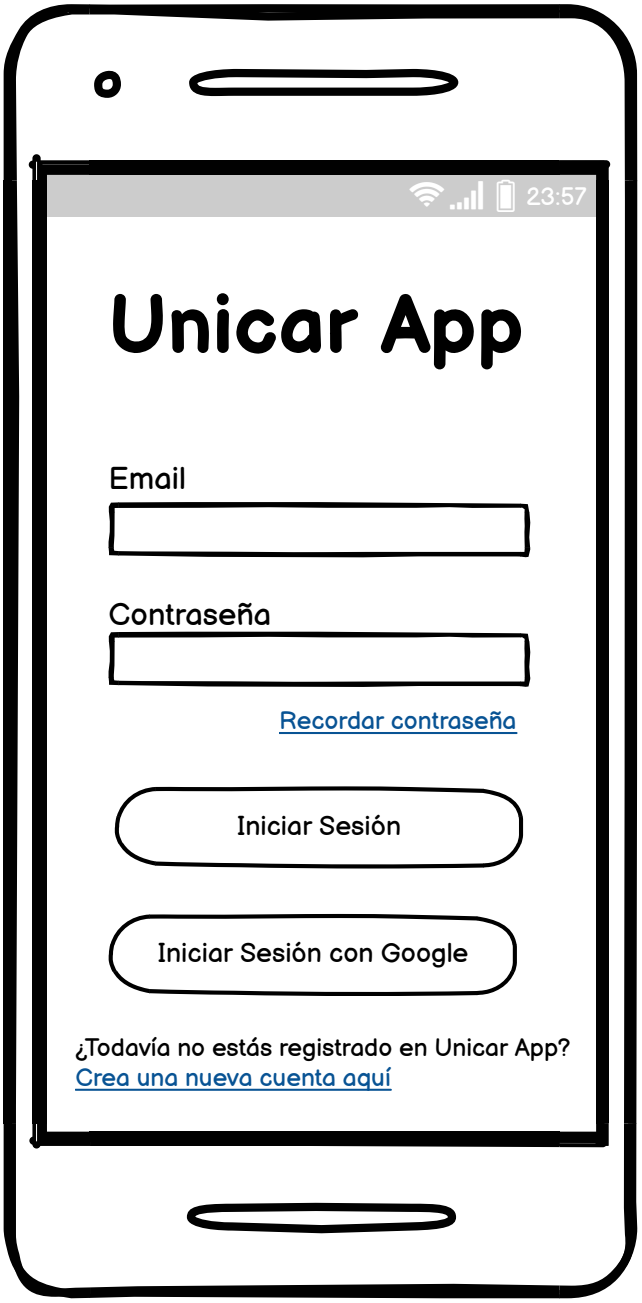
Inicio

Requisitos funcionales:

- RF1
- RF2

Descripción:

Inicio de sesión, inicio de sesión en Google.
Enlaces a recordar contraseña y registro.



Registro

Requisitos funcionales:

- RF1
- RF2

Descripción:

Para el registro tansolo será necesario el correo electrónico y una contraseña.

La verificación de la cuenta se realizará accedediendo a un enlace que enviado mediante email

Para finalizar el registro, en la página de mi perfil se tendrán que completar los demás datos:

Nombre de usuario, nombre, apellidos, localidad y universidad

Mis viajes

Requisitos funcionales:

- RF10
- RF11
- RF15
- RF16

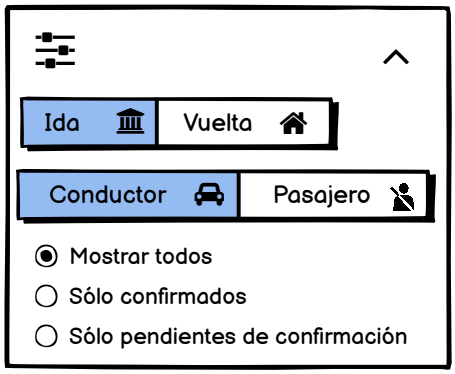
Descripción:

El usuario puede visualizar sus viajes pendientes ordenados por día.

El botón de "Mi calendario" va a la personalización de calendario.

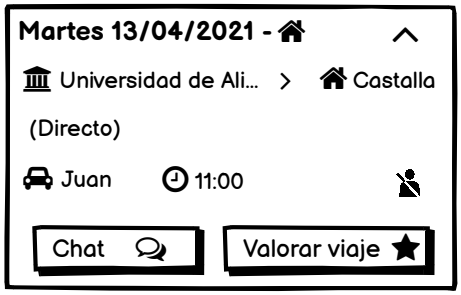
Si el usuario tiene el calendario personalizado el calendario, cada día muestra la hora de entrada y salida.

En cada día se indica el viaje de ida y vuelta. En caso de no tenerlos, el enlace lleva a la búsqueda



Viaje como conductor

Viaje pasado como pasajero



Viaje solicitado como pasajero

Viaje confirmado como pasajero

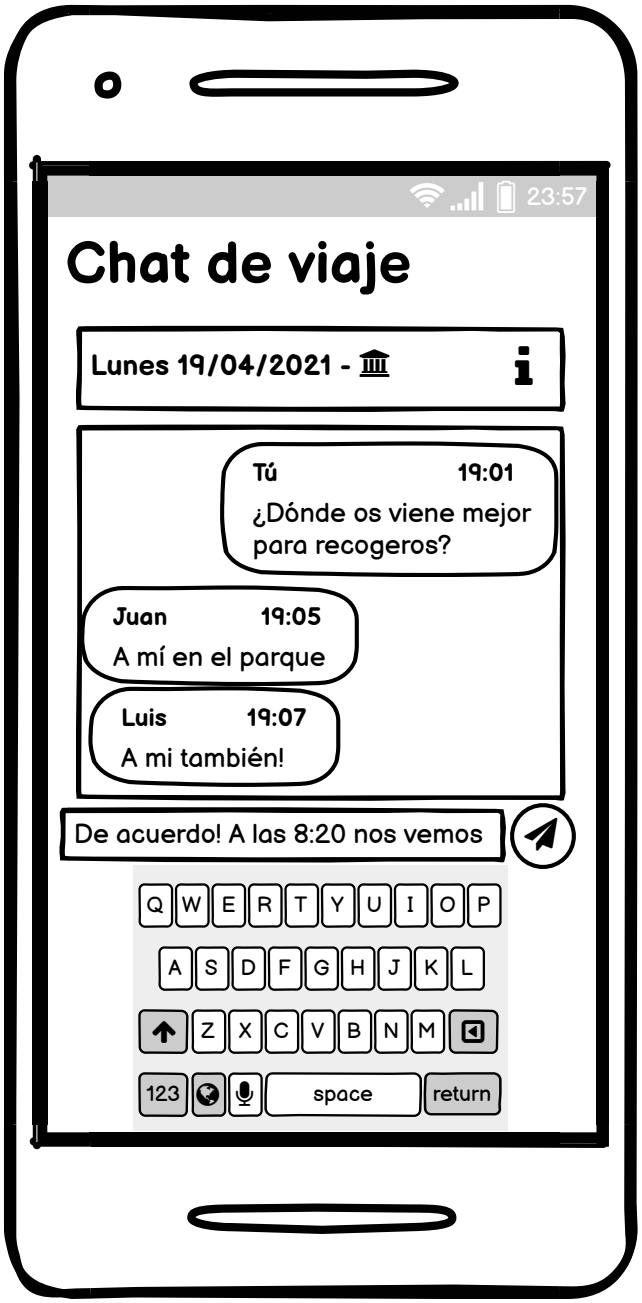
Chat de viaje

Requisitos funcionales:

- RF17

Descripción:

En el chat los usuarios podrán acordar la hora de salida y punto de encuentro para el viaje concreto



Solicitudes de viaje

Requisitos funcionales:

- RF8

Descripción:

En esta página se muestra brevemente la información del viaje y las solicitudes de plaza recibidas.

Estas podrán ser aceptadas o rechazadas

Valorar viaje

Requisitos funcionales:

- RF18

Descripción:

En esta página se muestra brevemente la información del viaje y los usuarios con los que se ha compartido coche

La valoración se realiza sobre el rol que ha tenido cada usuario durante el viaje

Buscar viajes

Requisitos funcionales:

- RF6

Descripción:

Muestra formulario con los filtros a aplicar

Los campos de Localidad y Universidad cambian de orden según el viaje sea de ida o vuelta.

Si el usuario tiene una localidad y/o una universidad establecida, estas serán las que se visualicen por defecto.

La fecha buscará el día exacto para el que se busque el viaje

En los viajes de ida, la hora a buscar será la hora de entrada a la universidad, por lo que se buscarán también viajes con horas previas similares

En los viajes de vuelta, la hora a buscar será la hora de salida de la universidad, por lo que se buscarán también viajes con horas posteriores similares

Viajes encontrados

Requisitos funcionales:

- RF6

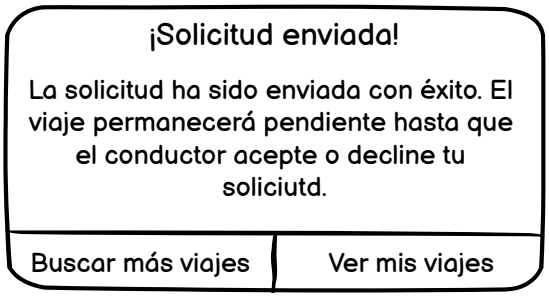
Descripción:

Se mostrarán los viajes que coinciden con los filtros especificados.

En la ficha de cada viaje se visualizará:
- Día de la semana - Fecha exacta - Sentido del viaje
- Origen > Destino
- Conductor (Valoración), Plazas ocupadas, Hora de llegada o salida

Cada uno de ellos se podrá desplegar para mostrar más información (Pasos intermedios) y el botón para solicitar la plaza

Para realizar una nueva búsqueda o modificar los filtros de la actual estará disponible el botón de buscar (lupa)



Publicar viaje

Requisitos funcionales:

- RF7

Descripción:

Se mostrará un formulario para introducir la información del viaje a publicar

Se podrá elegir entre ida y vuelta.

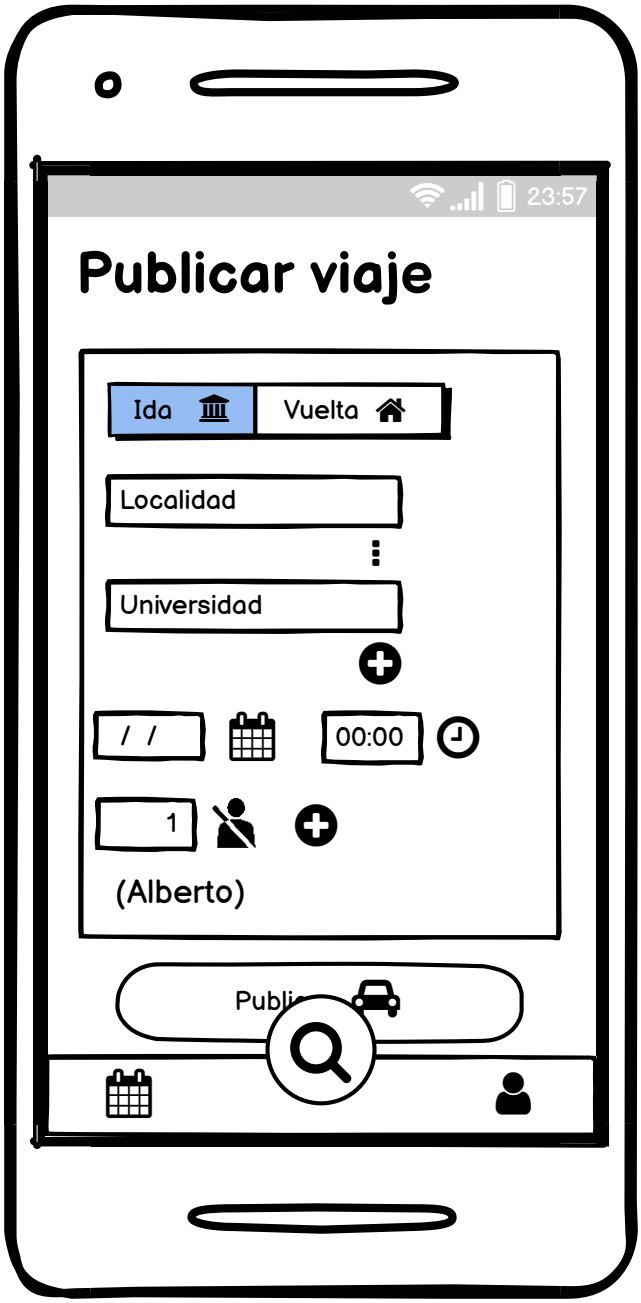
Según el sentido del viaje, la localidad y la universidad cambiarán su orden

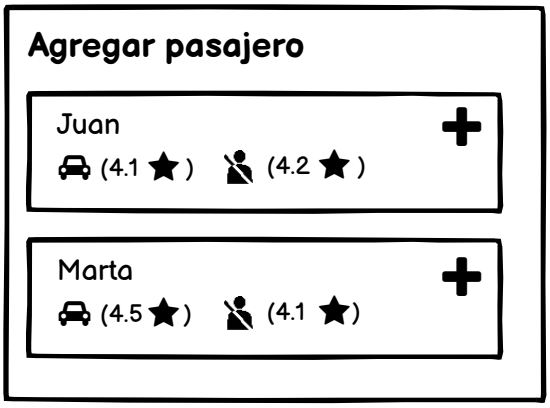
Se podrán añadir pasos intermedios con el botón +

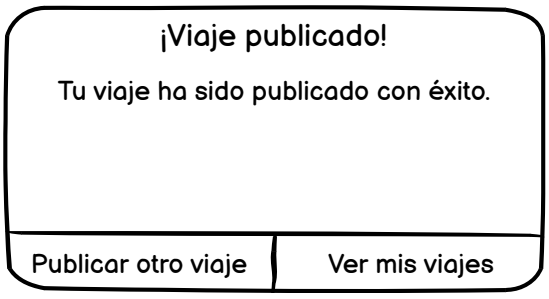
Se fijará una fecha y una hora de llegada o salida

Se introducirán las plazas disponibles

Se podrán añadir conocidos al viaje con el botón +







Mi calendario

Requisitos funcionales:

- RF9

Descripción:

Página para introducir o editar los horarios semanales.

El botón Importar calendario llevará a la página para importar el calendario

El botón Añadir viaje recurrente llevará a la página para añadir un viaje recurrente

El selector de semana se empleará como filtro para mostrar solamente los días de una semana

En cada día se muestra la hora de entrada y salida de clase, si el usuario las ha especificado.

La hora de entrada o salida se puede editar

Si un día no tiene clase o no se ha especificado la hora hora de entrada y salida pulsando en el enlace se mostrarán los campos para introducirlos.

Se podrá cancelar el horario previsto de un día pulsando en la X de la parte derecha

Publicar viaje recurrente

Requisitos funcionales:

- RF12
- RF14

Descripción:

Se mostrará un formulario para introducir la información del viaje recurrente a publicar

Se podrá elegir entre ida y vuelta.

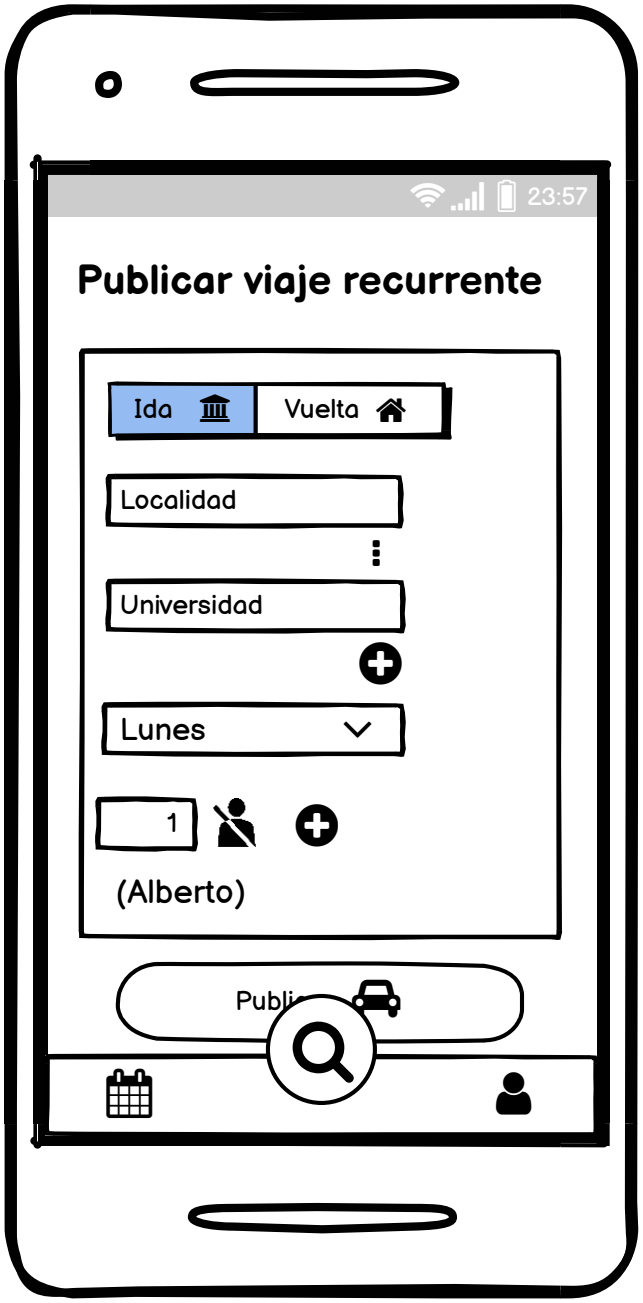
Según el sentido del viaje, la localidad y la universidad cambiarán su orden

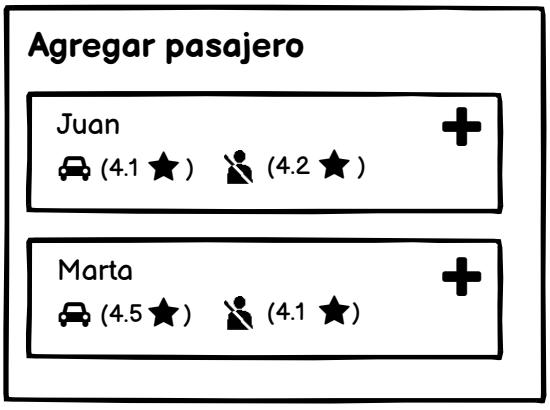
Se podrán añadir pasos intermedios con el botón +

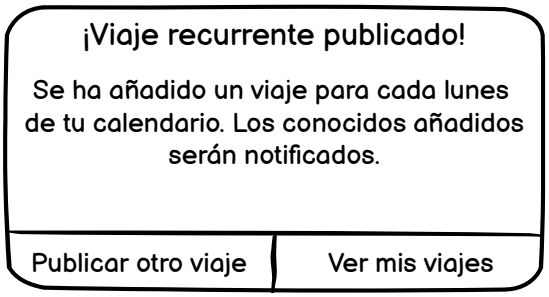
Se fijará un día de la semana. La hora será aquella establecida el calendario

Se introducirán las plazas disponibles

Se podrán añadir conocidos al viaje con el botón +







Mis viajes recurrentes

Requisitos funcionales:

- RF12

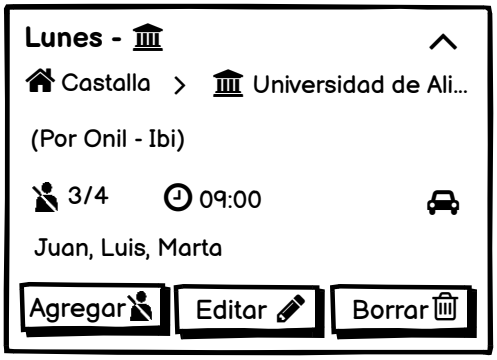
Descripción:

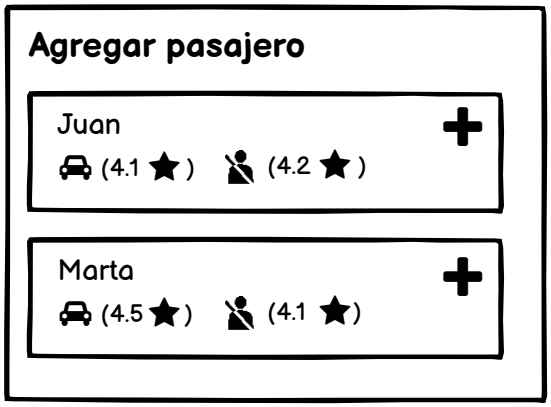
Se mostrará la lista de viajes recurrentes establecida por el usuario

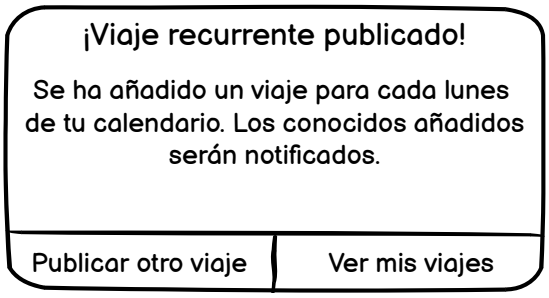
Se podrá añadir añadir un viaje recurrente a un día de la semana que no lo tenga o solicitar uno a un conocido. Cada enlace llevará a la sección correspondiente

Se podrán agregar pasajeros al viaje recurrente

Cada viaje recurrente podrá ser editado o borrado







Viajes recurrentes de conocidos

Requisitos funcionales:

- RF13

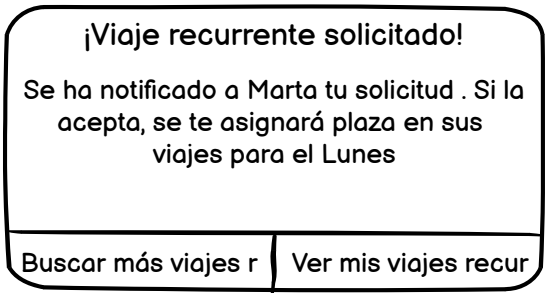
Descripción:

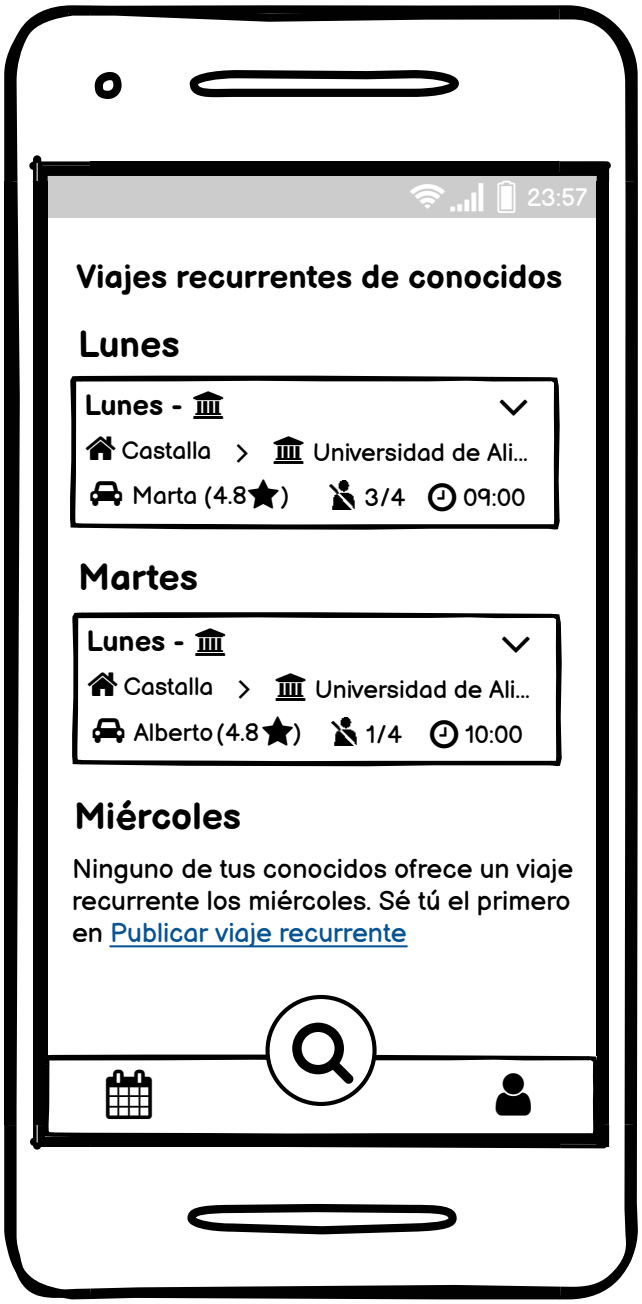
Se mostrará la lista de viajes recurrentes establecida por el usuario

Se podrá añadir añadir un viaje recurrente a un día de la semana que no lo tenga o solicitar uno a un conocido. Cada enlace llevará a la sección correspondiente

Se podrán agregar pasajeros al viaje recurrente

Cada viaje recurrente podrá ser editado o borrado





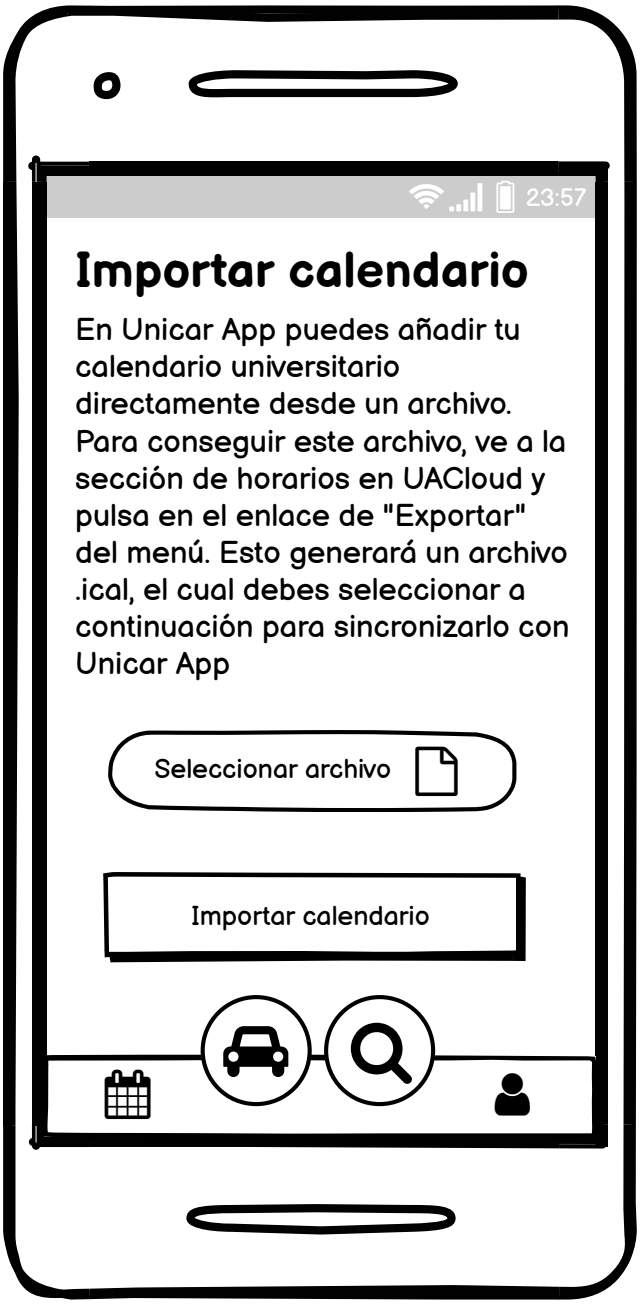
Importar calendario

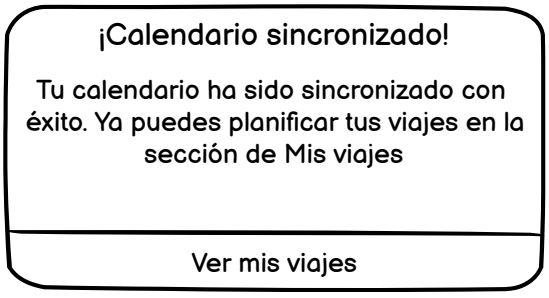
Requisitos funcionales:

- RF9

Descripción:

La página para importar el calendario consta de un selector de universidad una explicación de cómo obtener el archivo .ical necesario desde UACloud y de un selector de archivo para seleccionarlo y subirlo





Mi perfil

Requisitos funcionales:

- RF3

Descripción:

En la página de perfil el usuario tendrá una vista resumen de su perfil.

Tendrá la posibilidad de editar sus datos

Podrá acceder a la sección de conocidos

Podrá acceder a la sección de calendario

Editar perfil

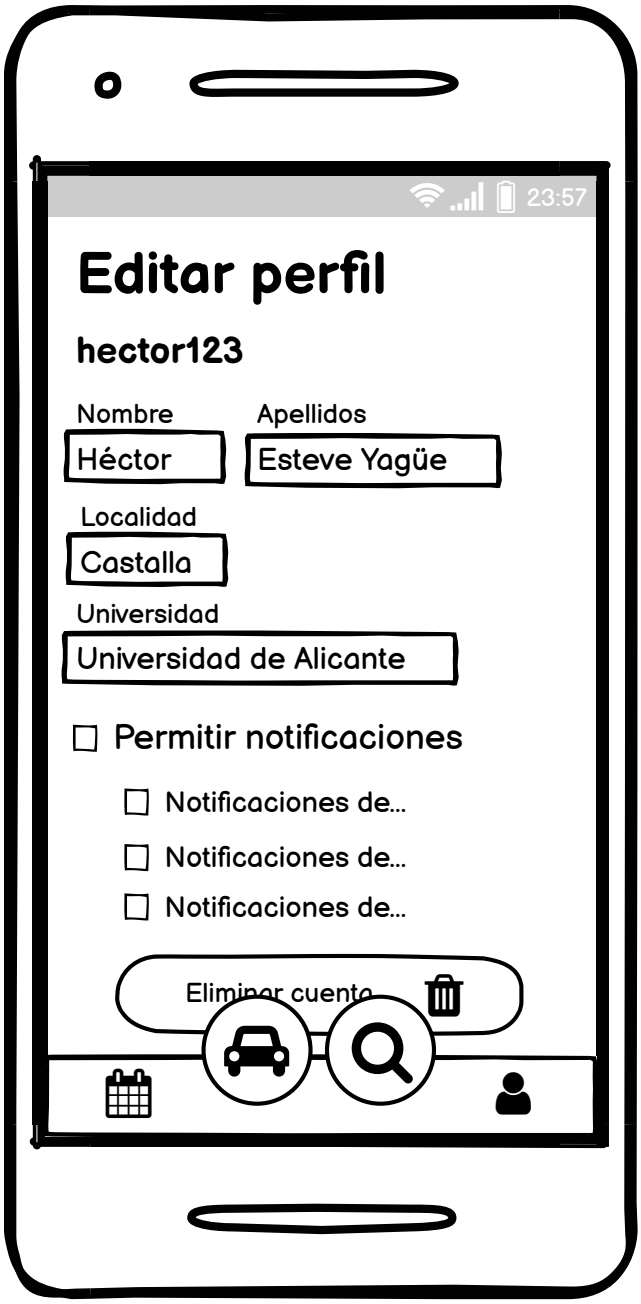
Requisitos funcionales:

- RF3
- RF19

Descripción:

En la página de editar perfil el usuario podrá modificar los datos de su perfil.

Podrá activar o desactivar las notificaciones y además elegirlas específicamente según el tipo de notificación



Mis conocidos

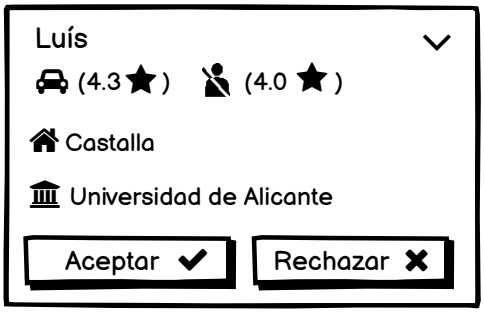
Requisitos funcionales:

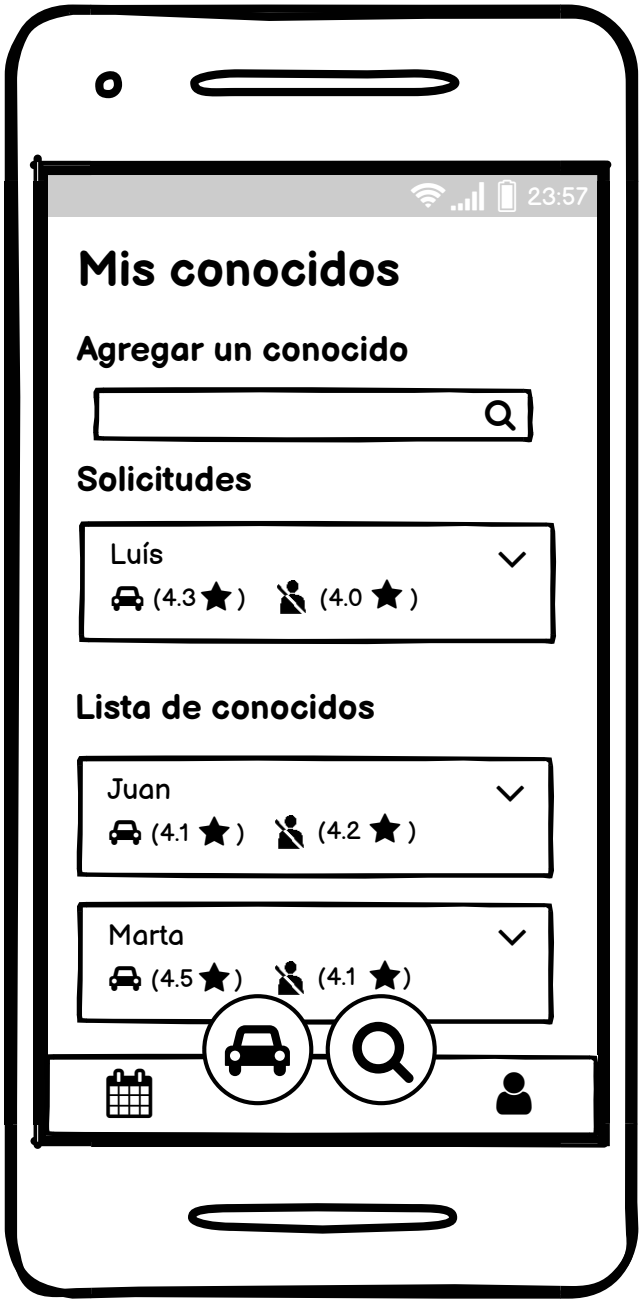
- RF5

Descripción:

En la página de conocidos, el usuario podrá visualizar la lista de conocidos, así como las peticiones recibidas para conocer a más usuarios.

Además podrá buscar usuarios





Buscar conocidos

Requisitos funcionales:

- RF5

Descripción:

En la página de buscar conocidos se mostrarán los usuarios cuyo nombre coincide o se asemeja al nombre introducido.

Se podrá solicitar el permiso para añadir a la lista de conocidos